Smart Autofill for Photos in Albums

Bachelor Project

of

Sjoerd P. van Egmond
#0400785

March 25, 2009
Period of research:
January - June 2007

Supervisor:
Dr. Ir. Fons J. Verbeek

LIACS, Leiden University
Niels Bohrweg 1, Leiden

Contents

Chapter 1

Introduction

1.1 General

This paper is the final result of my stage at Albumprinter where I did a research-project for a new algorithm to replace one of their existing algorithms that was to be improved because it lacked good user interaction.

1.2 Bachelor-Project

This project is a relatively large final project at the end of the bachelor. It is meant to have the student get to know how research is done by having him perform research himself or by having him take part in already existing research. Another goal is to have the student use the knowledge and practical experience he has gained throughout his bachelor-years. And last but not least it encourages the student to be more independant and responsible.

1.3 Albumprinter

The next few paragraphs will describe Albumprinter, the company at whose request this research has been done.

Albumprinter is a software company that develops a program that enables users to create their own virtual photo books, photo calendars and other photo products. The software then allows the user to upload their virtual product, which gets printed and posted when it is paid. The result is a high-quality photo product home-delivered.

Since the actual start of selling photo albums since 2003 Albumprinter has grown to become the market leader of printing digital photo albums in The Netherlands, as well as in Europe a couple of years later. Their program has won the 'Best Product'-award of several magazines and papers since.

1.3.1 The Products

Albumprinters most highly evaluated products are its photo books, used for photos of a vacation, party, marriage or any other theme the user might think

of. Users can completely customize their product to fit their needs: by changing the lay-out, adding text and backgrounds and creating a photo-cover to enjoy creating their own photo albums.

Besides photo books Albumprinter has a range of other products. For example there are quite a bit of different calendars (e.g. yearcalendars and birthcalendars), but there are also posters, agenda's and more.

1.3.2 The Program

At the moment Albumprinter offers a free program on their website [Alb]. This program enables the user to choose one of the products described above, which is the first step in creating their product. When the type of product has been chosen the user can completely customize their product by adding or changing photos, text, backgrounds and more. The next step for the user, when the product is finished, is to order the product by having the program upload it and then entering his or her personal information for payment and delivery. The last step is to pay for the product and when the payment has been received by Albumprinter the product will be printed and sent to the user.

1.4 The Problem

The software described earlier already has a lot of functionality, like a photo editor, a text editor, possibilities to fit the product to the users personal needs and a function to automatically add a large group of photos to the product, the so-called autofill. Unfortunately not all functionality is user-friendly, and especially previously mentioned autofill would benefit from a newly developed way of working.

The autofill in its current form doesn't look at the photos before they are added to the product. It just takes a pre-defined product and fills the photos in this product in the order in which they are delivered, on the places pre-defined by the product. But this also has the effect that groups of photos can be cut in two, a low-quality photo is inserted at a very important location or vice versa.

1.5 The Assignment

The assignment given by Albumprinter is to find a solution for previous mentioned problem by researching a new, smart and user-friendly algorithm that will work within boundaries still to be given. These boundaries will ensure the algorithm will be usable in their product, by giving some practical preconditions.

Chapter 2

Assignment, Materials and Methods

2.1 The Assignment

As mentioned in the previous chapter the assignment covers creating a new and improved algorithm for the autofill. Albumprinter wants an extendable algorithm that is able to divide the photos over the pages in a custom way. The algorithm has to be extendable in a way to allow easy adding of extra functionality that will enable the algorithm to perform better. The algorithm has to find an optimal division of groups of photos to put on the pages of a product. The order of those photos is fixed and meta-information from those photos can be obtained or delivered as input. The output from the algorithm should be a custom-filled product optimized for the chosen photos, or better said: it should deliver the information with which the editor can create the product.

Limitations with respect to this assignment come in two flavours, run-time boundaries for the algorithm and build-time limitations to ensure compatibility with the editor.

One of the run-time boundaries basically asks for a method to send information about the type of product chosen by the customer to the algorithm, so that the limitations of that product can be enforced. Another run-time boundary, probably the most important boundary, says the speed of the algorithm should be neglectable by the user. Meaning that if the algorithm gets called the user should not have the feeling of waiting for the algorithm to finish, since the step after the autofill, which is done by the editor, already creates a small but sometimes significant waiting time for the user.

The build-time limitations are the fact that development cannot take place in the main product, and when finished communication with the editor should be relatively easy. As said previously the algorithm should be written in such a way as to allow for easy extensibility, especially for adding more meta-information about the product or the photos, so when new information about the photos gets known and is added to the input only minor adjustments to the algorithm are needed. Preferable it also should be possible to extend the assignment with a minimum of work to also enable it to be reachable through the internet, but

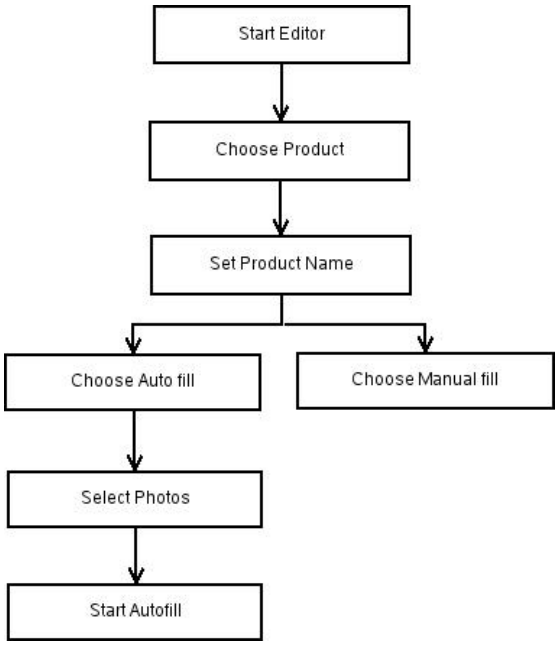

Figure 2.1: Flow-chart of current functionality

this is not explicitly asked for in the assignment.

2.2 Analysis: current functionality

2.2.1 Albumprinter Editor in use

In the current editor the user will interact with the autofill only by choosing the photos that need to be filled in. See Figure 2.1 for a flow-chart.

When the editor is started the user will be given the option to choose a product. There are several products available in the editor from which he can pick his product. When a product has been chosen the next step for the user is to fill in the name of his product and choose to manually or automatically fill the product. If the manual option is chosen the autofill will not be executed, but if automatic is chosen the next step for the user will come up, which is a screen in which he can choose his photos. The user can choose from zero photos to be filled to double the amount of photos that can fit into the product from any location on the computer. On this screen it is shown how many photos the editor expects. There is also an option that enables the user to select the number of pages to fill, so he can fill in the last pages himself. When he is finished picking his photos he can continue and now the autofill will do his work. When less photos than needed are chosen the last part of the product will remain empty, and when more photos than needed are chosen the remaining photos that do not fit will be kept out. The functionality to be able to insert more photos than needed arrives from the possibility that photos are rejected because their quality is too low, so another photo is needed to fill the rejected photo's gap.

2.2.2 Autofill

In its current form the autofill works two ways, both from the users side where the user selects his photos as from the side of the product. From the side of the product there is a pre-defined distribution of the photos over the pages. Meaning that when the user chooses to load a new product, without adding any photos, he will see an empty photo product but with pre-defined locations where a photo can be inserted. From the side of the user a number of photos is chosen, in a certain order, which have to be inserted in the product. Now when the autofill executes his function it will just insert the photos in the given order into the product.

A consequence of this operation is that the dimension of the photos is not taken into account. An undesired effect of this is that a landscape-oriënted photo might be inserted on a location where a portrait-oriënted photo is expected or vice versa. When this happens a large part of the photo will be cut off to fit it on the pre-defined location. Furthermore large, high quality photos might be put in a very small location, while low quality photos might be put on an entire page. Moreover, if a user has decided on all the photos in his book and the amount of photos was incorrect compared to the amount needed by the product, it is necessary to manually add or remove pages with photos.

2.3 Requirements: future functionality

2.3.1 Albumprinter Editor in use

For the user not much will change by integrating a new autofill algorithm, see Figure 2.2. A choice is made to let the autofill have no more influence on the original operating then needed, none if possible. There is absolutely no different functionality needed before the editor arrives at the autofill-screen. On this screen two minor changes will have to be made: first the option to allow for less pages to be filled should be removed, and second the text indicating the amount of photos needed is not correct functionality anymore.

Since the autofill creates a custom product dependent on the selected photos it is not possible anymore to have any empty pages at the end of the product. The user has to choose at least a minimum amount of photos to be filled. Therefore the text that describes how many photos should be filled should be changed to a description about how many photos minimally and maximally have to be chosen for the autofill to work. In the new functionality it is not possible to insert only a few photos at the start of the product and fill the rest by hand.

2.3.2 Autofill

There are two different kinds of requirements that are applicable to this algorithm. The familiar ones that describe what it has to do, i.e. in what way the algorithm translates the input into usable output. The other type of requirements describe dependencies with the program in which it will have to be integrated.

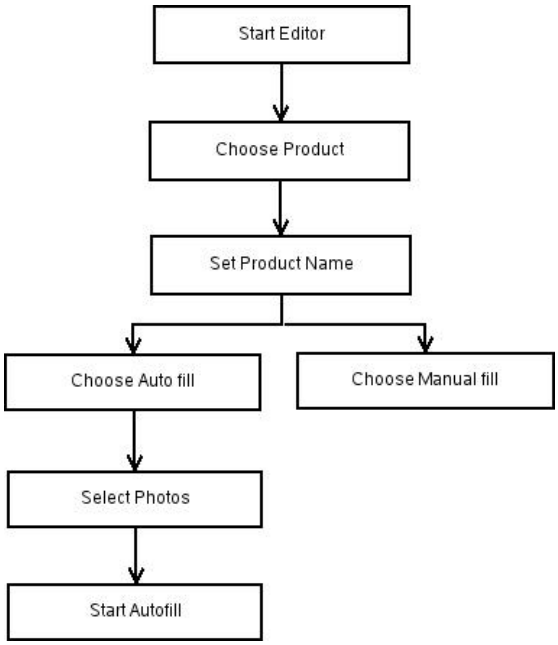

Figure 2.2: Flow-chart of new functionality

Algorithm

The exact form of the new algorithm is not known at this point. Most of the requirement known up-front have been described before in Chapter 2.1, but other requirements or limitations may be found during development of the actual algorithm.

Implementation

Since the implementation of the algorithm will have to work in the album editor integration of the algorithm and the editor is of a very high importance. The reason for some of the implementation-choices that are made are because of the build-time-limitations mentioned before [Chapter 2.1]. Detailed descriptions of the implementation-choices will be given in the materials-section [Chapter 2.4], but the implementation will be done as a DLL coded in Delphi with an XML-file as input and as output. The XML-file will follow the RDF-format [W3C].

2.4 Materials

Because the algorithm has to be integrated into an existing project the choice for a language is very limited. Since the development of the algorithm will be separated from the development of the entire editor it is favourable to develop it as a dynamically linked library, a DLL. The choice for a language is now limited to a language able to produce DLLs. But since it has to be integrated with the editor project, which is written in Delphi, Delphi is the language chosen to implement the algorithm in since it is also able to product DLLs.

For input and output of the DLL the choice fell upon using files written in XML, since large amounts of descriptions of the product and the photos need to be given to the algorithm. Because the only thing needed to be send to the DLL is metadata, the XML-variant of choice is RDF (Resource Description Framework) because it has been especially designed to describe metadata and relations between different items.

2.4.1 Delphi

The programming language chosen is Delphi, because the new autofill has to be integrated with a program written in Delphi. And because Delphi itself is quite a powerful language. It has roots as old as other major programming languages because it is derived from Pascal, and it is a very complete and adult language. The environment available to create Delphi projects is Turbo Delphi [Cod], a free IDE for Delphi development.

2.4.2 DLL

Since the autofill has to be developed outside from the main application, a way had to be found to easily integrate it within the Albumprinter application. Several options are available to do this, but the two that come up best are to create it as a module and later add it to the applications own code-base, or create it separately as an extern module and have the application call it externly as a DLL [Wika].

The choice for developing it as a DLL was because it had more benefits than creating it as a module and inserting it in the Albumprinter application. This way further development will always remain easy and separate, is it possible to create a web-service from it and last but not least only one codebase will have to be in existence at any given time.

2.4.3 RDF

Since the choice was made to have the autofill as an external module a way had to be found to transport all the meta-information needed by the autofill to it, and the results would have to be transported back. To transport such massive amounts of information about different items it is useful to try and combine it in one information-set, which in this case was chosen to be a file. But just sending a file with information in it isn't useful, what the information is about and the order of the information (in case of the photos) is also important. Since only meta-information is supposed to be sent and the order-relations of the photos should also be preserved a sublanguage of XML found its way to the project, namely RDF [W3C]. RDF is a language explicitly created to describe meta-information and relations between the several parts of that meta-information, while XML cannot guarantee to keep the order of the photos and therefore 'just XML' would not have sufficed.

Examples of the meta-information that need to be sent are limitations of the chosen photo-product, descriptions of the photos and the order in which the photos are sent.

2.5 Methods

About the method of research I can be relatively short, first and foremost I will follow the standard software-development method. First set up the requirements [Chapter 2.1], then find a way to solve the problem on paper and discuss the way in which to solve it with the other people involved and get their opinions. Some of the choices of the materials [Chapter 2.4] will be made by discussing the available options with the other people involved.

After the decisions and the paper-solution it is time to create a working prototype to check whether the implementation is viable and correct. If for some reason a problem arises, either because the implementation is too complex or because the prototype demonstrates a problem concerning the requirements, it is time to go back to the drawing-boards and rewrite the paper-solution keeping in mind the problem that was just showed by the prototype. This method is repeated until the arrival of a viable implementation. At the moment a viable prototype is found it is time for a full working version to be created, tested and integrated within the Albumprinter application.

Chapter 3

Implementation

3.1 Introduction

In this chapter the implementation of the algorithm will be described. Every part that fulfills (part of) an essential execution will be explained. Furthermore a description will be given of the way testing is performed.

3.2 Algorithm Description

3.2.1 Short overview

The complete algorithm can be split up into four parts, of which two will be described together: input and output. The other two parts generate a rating for the photos and then distribute the photos. Input is in the form of an XML-file, or more precise, an RDF-file. It contains information about the photos and the product. The photo rating is calculated using the information from the input. The more information can be found in the input the more complex the photo rating can become. To distribute the photos the previously generated photo rating is used, and this part is the heart and brains of the algorithm. It performs a smart-search, the first distribution to fit will also be the best distribution that can be found. So when a distribution has been found the algorithm can stop and continue with the last step. The last step is creating output in a form understandable for other programs.

3.2.2 Input & Output

For the input and output the choice fell upon using RDF-files as data carriers. There were several reasons for this choice, but the main reason was to encapsulate the algorithm within its module as much as possible. It ensures that no matter the amount of data needed to be transferred only one function call needs to be executed. Further development also does not change the call that needs to be made, which is a situation preferred from the point of object-oriënted-programming.

3.2.3 Photo Rating

The ratings of the photos are the basis for the entire further calculation. But this also means that even before starting on the actual algorithm some calculations have to be done to ensure an appropriate rating for the photos.

The ratings are calculated by finding the relations for every photo to the photos next to it. So every photo has initially two ratings: ratings respectively indicating the relation with the previous photo and the next photo. The ratings get changed according to several properties concerning the photo or the relation between two photos. When all properties have been handled the photo now has two ratings, to find the final rating r_f for a photo the following formula is used, with r_1 and r_2 being the ratings for the relation to the previous respectively the next photo.

$$r_f = 1/(r_1 + r_2) \tag{3.1}$$

During calculation both r_1 and r_2 are bound within the range of 0 and 1, to ensure a certain range between 'the perfect location to divide the photos' and 'the perfect couple of photos for one page'. When r_1 or r_2 is 0 it means that there is no connection between the photos so it is preferred to split them up over two pages, while if one of them has the value 1 it means that the connection to the photo on that side is so good they should be on the same page.

With r_1 and r_2 within this range there is also one implicit boundary for r_f, namely a minimum of 0.5, reached if both r_1 and r_2 have the value 1. The maximum boundary is explicitly introduced to keep the ratings within a usable reach from each other and also eliminating overflow- or division-by-zero-exceptions. The used maximum is 10, which will be more than enough if looked at what the values of r_1 and r_2 have to be together to reach 10: namely a maximum of 0.1, which will already indicate this photo is especially usable as inserting it on a page with no other photos.

The basis for how much every property should be able to change the ratings is saved within constant variables. These constants indicate how much a property should be able to change the rating if the ideal property is reached. The values of every property for the photos should first be changed to be within the range of 0 and 1 after which they should be multiplied with the appropriate constant and added to the ratings. To be sure numbers are not only added but sometimes also subtracted some negative constants are used, or the factor with which it is multiplied is set to a negative number (for some properties a factor is chosen depending on the ranges instead of 'normalization'). If for example a property is allowed to change the photorating with maximal 40% and there is a photo that has this property with value 0.75 it changes the rating with 0.30.

The last thing to mention concerning the ratings is that every property updates exactly two ratings, but certain ratings concerning one photo update r_1 and r_2 of the photo, while other properties update two ratings but r_2 from photo X and r_1 from the next photo.

3.2.4 Distributing the Photos

This is the hardest part and the core of the entire algorithm. In this part of the algorithm an actual distribution is calculated for the photos. A really short description: it is a breadth-first-search algorithm which builds a tree during execution. The tree is an abstract form containing the distribution of the photos

over the pages. The algorithm has a method to return to an earlier point if it seems impossible to finish the tree in this way. It stops when it has found an entire tree, and also has a mechanism preventing it from trying to build a tree that was earlier created but already dismissed to prevent endless loops.

There are several parts that need to be explained in detail, which are the picking-function (which photos will be put on one page), how does the tree work, in which order is the tree expanded and how is it ensured the algorithm will not fail when it has made a mistake.

Picking the photos

As described before at this moment every photo has a certain rating, ranging from 0.5 to 10. Another thing already known is the minimum and maximum amount of photos per page. This is combined to create another rating composed of the ratings from a certain photo and a certain number of photos following it. This means that when a page is allowed 1 through 3 photos every photo will have its own rating, a rating composed of itself and the next photo divided by two and a rating composed of itself and the next two photos divided by three. The picking-function operates on a certain range of photos, but this does not change the way it works because it just finds the highest rating and checks if the chosen photos and location of the page will generate no problems.

There is only one more thing to say about the picking-function. Since there is a variable indicating how much the pages can differ from the average and only a certain number of pages is allowed the pages will already have been calculated, or better said, the number of pages allowed to contain a certain number of photos is already known. This makes sure that the picking function does not always picks a single photo but has to pick a number of photos for which still a page is left. If this restriction wasn't present a single photo would always be picked, because the rating of the highest-rated photo will always be higher than when this rating is combined with another rating, even if this was the second highest rating

Abstract vision of the tree

The root node of the tree will initially contain the entire range of photos that has to be filled. Whenever a page has been found the node in which the search took place will change to contain only the photos for the found page and the node will be marked fixed ('this is a page'). Following this zero, one or two children will be created for this node containing the left-over photos from the range in which the algorithm could pick its photos. If there were photos left 'before' the picked photos a left child will be created. This is done respectively for the right side with photos 'after' the picked photos. To explain before and after: the photos are inserted in a list and will be assigned a number. This is done to ensure the order given by the user is kept.

When all nodes are fixed a distribution for the photos has been found, so the algorithm can stop and the result can be found by doing a left-root-right-walk on the nodes.

Tree expansion

The tree is expanded in a breadth-first-search way, expanding all the not-fixed nodes on a certain level consecutively. This ensures that above a certain level all nodes are fixed. The reason for this type of expansion and not the easier depth-first-search is because in general the pages with fewer photos on them will be picked first, because fewer photos usually means a higher rating. So if depth-first-search was used the first pages of a product would contain the lowest amount of photos per page, while the pages at the end would contain the highest amount of photos per page. With breadth-first-search this still happens because breadth-first-search starts also on one side of the tree, but it happens on a much smaller scale.

Rollback

Since it is possible for the algorithm to generate a partial tree that cannot be expanded anymore but that still has photos left it is essential to be able to go back towards a smaller partial tree that can (possibly) continue where the previous tree failed. But besides being able to get back to such a smaller partial tree it is as important to be able to recognize the previous mistake and take another path that does not lead to the same tree again. This even has to be ensured for all previous failed trees if it takes a couple of tries before the right choice is made.

The rollback chooses the parent of the node in which no possible expansion (or fixing the node as a leaf without children) is possible. This is the logical choice if it was a depth-first-search algorithm, because with recursion a stack would be implicitly created and saved and the algorithm could just go back, return that it failed and have his calling function try again. With breadth-first-search this is not as easy, since no stack is saved implicitly and re-stepping all operations done would require massive amounts of bookkeeping everything that has been done. Instead the chosen option was to create an explicit stack in which it is possible to return an arbitrary state back into the algorithm with only a minor adjustment in the state indicating which photos were last picked and failed and thus should not be picked in that combination again.

Shortly described every node has a number x, with the root starting with zero, for which the left child gets number $(x * 2) + 1$ and the right child gets number $(x * 2) + 2$. This ensures that no node ever gets the same number as another [Wikb], while it is possible to calculate the number of the parent from $(x - 1)/2$. If a rollback on a node needs to be done: find the number of the parent-node b, find its saved state, remove the old information the algorithm used, insert the found saved state and clean up the stack to contain only states for nodes up to b.

3.3 Testing & Prototyping

3.3.1 Testgroup

The initial tests were done by myself and the developers of the album-software because this DLL would be linked to the program and they themselves wanted

to see that it worked correctly and within the given boundaries before implementing it.

The second testing-round was done by the QC-team (Quality Control) of Albumprinter as part of the quality-check of the new version of the album-software.

After an OK from QC the software is ready to be set live and be distributed to the end-users, but usually the editor can be tested by other employees first and they sometimes make a useful remark about the functionality. Since this round is the first in which non-developers see the program, it can be considered a preliminary to live.

And last the editor is set live, can be downloaded by the users and they can play with it.

3.3.2 Test Method

For the first tests I have written a small application which functionality is to load a file with a description of photos, call the DLL with this file and show the results on the screen. This method was chosen to give the developers enough of an idea of how well the project works. This also allows them to play with it by changing the file with the photo-descriptions and see what changes occur with different photo-sets.

The second method of testing is done by the QC-team, and they methodically try dozens of photo-combinations to see if they can break the program. As a result they also see a lot of output of the algorithm and can give their feed-back to how well it works.

The last method is to have users create a book and ask what they thought of how the new autofill-algorithm fits within the program and how content they were with the result it delivered.

Chapter 4

Results & Analysis

4.1 Results

The results of this project come in the form of two delivered DLLs and a module to integrate it with the Albumprinter program. The first DLL is compiled to be used by a stand-alone executable, and the second has been encapsulated with an ISAPI-interface to allow it to be called as an internet-service. As of now the DLL for the stand-alone executable has been integrated with the software [Alb], but the ISAPI-DLL is not yet in use.

But these are only the final implementations of the program. Now for the results and the analysis of the project itself.

4.2 The Algorithm

The algorithm itself works within the boundaries given. The output of the algorithm conforms to the restrictions described in the input, the order of the photos remains as given, all photos are used, the distribution is as good as possible and last but not least the speed of the algorithm clearly falls within the boundaries even for large sets of photos. Take a look at Appendix A.1 for an example of the result.

A description of the results of how well the algorithm conforms to the restrictions given in the input can be very brief. After extensive testing under the restrictions there hasn't been one single occasion in which the output went beyond its given boundaries, neither before integrating it with the Albumprinter software, nor afterwards when it was used within the software.

As for the speed results I have a little bit more to say, especially about what the problems were and how they were overcome. As you can see in Table 4.1 and Figure 4.1 the time it takes the algorithm to find a solution is almost neglectable and can be described as completely neglectable when compared to the time it takes the software to build the product after the autofill has given its result. Especially for commonly used numbers of photos, up to 300, the time is neglectable. But this result wasn't obtained on the first try.

The early versions of the algorithm became slower and slower as more photos were added to the input-set, and this effect went exponentially because the first versions were implemented using a depth-first-search algorithm which had

# Photos	Tested Speed(milliseconds)
10	44
20	79
50	234
100	551
300	2352
500	2737
1000	6194

Table 4.1: Speed per Photos

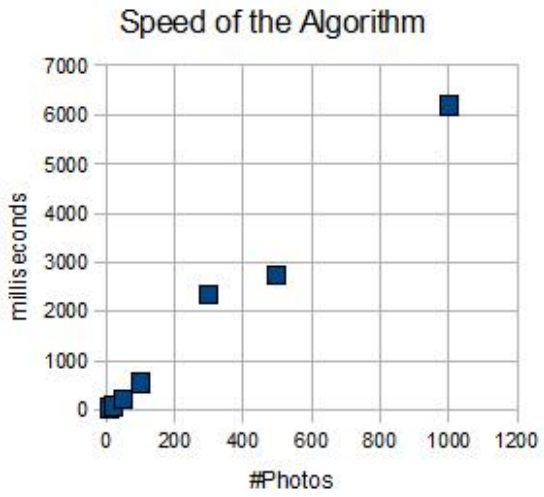

Figure 4.1: Results of the speed of the Algorithm

to rollback regularly and every added photo increased the chance that more rollbacks had to be made. For small photosets this version could be viewed as an optimal implementation, because it finds the best solution in an acceptable time. But the moment the photoset increased to a number that did not find a solution in a neglectable timespan it was necessary to find another solution.

The second set of prototypes used a breadth-first-search (BFS) algorithm as opposed to the depth-first-search (DFS) ones from before. The idea was that if the DFS algorithm would take too long to recognize that a certain branch would never find a solution while it was busy trying to find it a lot of time would be wasted. So using BFS the algorithm should be able to faster recognize that a certain branch would die so it wouldn't waste all that effort into trying to optimize a small branch over and over again. And as expected the BFS algorithm was remarkably faster than the DFS one on the same sets. Testing this new version however saw some new problems arise. It could handle more photos at once, but there was still a limit to the amount of photos it could handle in a neglectable time. Another problem of an entirely different nature is that the distribution of the pages over the album was askew. This problem could occur because breadth-first-search works from one side of the list to the other, and always starts at the same side. In this case at the front of the line of photos,

and the effect of this was that at the start of the book more of the pages with an amount of photos less than average are inserted while at the end the pages with more photos than average are inserted. While some users might prefer such a distribution because it could emphasize the attention of viewers to the first pages, but for a good and fair distribution this solution isn't satisfactory.

The third solution is the final solution, in which the two abovementioned problems were tackled. The problem with the skewed distribution was tackled first, because that was the more difficult solution and it required a massive rewrite of the algorithm. Because the problem had to be solved from two sides, on one side the photos and on the other the pages, a normal breadth-first-search algorithm would not suffice, so a small extra element was introduced. When a page was chosen to be inserted on a certain spot the photos left and/or right of it still would have to be distributed. The second solution kept the pages in one pool to be drawn from, but now the pool would be split into separate pools (valid pools, so if a certain number of photos had to be filled the total amount of photos in the pool would be the same). If we now arrive in a situation where a page doesn't fit where it was assigned a rollback has to be made to the point where the pool was last divided. So a small element from depth-first-search has to be introduced, namely its depth-based stack. The rewrite of the algorithm was needed to be able to write the manual depth-based stack in a breadth-first-search algorithm. The speed-issue was easier to fix, because the speed decreased exponentially with more photos. So if I would run the algorithm twice with only half the photos it would finish much faster. The only tricky part concerning splitting up the photos is that it can be possible to split it up right between two photos you want to keep together. So splitting up cannot be done exactly with a certain number of photos, but it can be done by searching a bit before that amount of photos for a spot where it is likely that a page-break would have been inserted. The amount of photos that has been chosen to keep within one block is one hundred, because that is fast but also allows for enough variation of pages within the block itself. The result for 300 photos was a speed-up from ± 30 seconds to ± 1 second.

Chapter 5

Discussion & Conclusions

5.1 User Experiences

The results gained from evaluating the algorithm through user satisfaction had a mixed content. But let's start with the negative feedback and try to explain where that came from. The negative feedback came from some of my colleagues within Albumprinter when the autofill project was just integrated with the Albumprinter Editor. Some of them had expected it to blindly follow their input about which photo was important and which not, but the algorithm was, and is still, limited to the meta-properties it arrives about the book, and if those properties describe for example that every page should have a minimum of two photos on it the algorithm isn't able to distribute only one photo on a page. But their thinking that the algorithm could do that made their expectance much to high. After explaining the reasonings behind the algorithms behaviour the negative feedback from that side stopped.

On the other hand though the reactions from end-users were very positive, and mostly only positive. Though some of the positive feedback might have to do with the new way in which the book was created, this still was an effect from the new autofill algorithm. The improvement from the old algorithm was seen as a drastical change, but in a positive way. Even when later the option to rate your photos was removed because it did not fit in the direction the editor should go (it took too much time) the new autofill was still perceived as much improved because of a small randomized component. This gave the autofill a playful component which the end-users very much liked, even though they had to finalize their books themselves it gave them ideas on how to start.

5.2 Conclusion

In the end the core of the project has been finished within the given limitations, and it is received by Albumprinter as a successful project. Another part of the project that has been well-received was the ease with which it can be expanded with new ratings for the photos. Inserting another variable for the photos does not in any way change the core of the algorithm, nor does it touch the functions with the other variables. In another section some more option for further development will be discussed [Chapter 5.3].

The research for this project was set up really practical, characterized by the fact that some time and development went into solutions that were not viable, but it was still useful to learn what can and can't be done when restarting from a point in the past. On the other side however it was interesting to see how the algorithm developed over time and what some of the problems were, how they came to be and how to get rid of them again.

5.3 Further Development

The core of the algorithm does what it is supposed to do, and although it is not done to call *anything* finished it is better to focus on other parts that obviously can be improved. Therefore further development of this project is best done in the field of the photo variables. As of now there are a couple of variables that are used, namely an overall rating of the photo given by the user, a quality rating for the photo automatically calculated, and two ratings indicating similarities with the photos directly before and after it. But other possible ratings could be what is on the photo, so for example close-ups of faces could be recognized, or a set of photos with horizons could be grouped together on one page.

Another course of further development for example is not directly about the photos, but more about how the distribution is calculated. For example somebody could take a look at creating themes for distributions, and the algorithm should take this into account and try to distribute the photos according to a pre-defined theme which could emphasize certain photo variables more or less.

Bibliography

[Alb] Albumprinter.com. Albumprinter-website. http://www.albumprinter.com.

[Cod] CodeGear. Turbo delphi. http://www.turboexplorer.com/delphi.

[W3C] W3C. Resource description framework. http://www.w3.org/RDF/.

[Wika] Wikipedia. About dlls. http://en.wikipedia.org/wiki/Dynamic-link_library.

[Wikb] Wikipedia. Methods for storing binary trees. http://en.wikipedia.org/wiki/Binary_tree#Methods_for_storing_binary_trees.

Appendix A

Examples

A.1 Results

Figure A.1 shows us one of the possible results for performing the algorithm containing a list of landscaped photos with 4 portrait oriënted photos between them.

Figure A.1: Resulting layout of the Algorithm