A Versatile Open-Source Photomosaic Maker

A Technical Report submitted to the Department of Computer Science

Presented to the Faculty of the School of Engineering and Applied Science

University of Virginia • Charlottesville, Virginia

In Partial Fulfillment of the Requirements for the Degree

Bachelor of Science, School of Engineering

Hanzhi Zhou

Spring, 2022.

Daniel G. Graham, Department of Computer Science

Rosanne Vrugtman, Department of Computer Science

A Versatile Open-Source Photomosaic Maker

CS4991 Capstone Report, 2021

Hanzhi Zhou
Department of Computer Science
University of Virginia
School of Engineering and Applied Science
Charlottesville, Virginia, USA
hz2zz@virginia.edu

ABSTRACT

Photomosaic is formed by tiling many small images (tiles) to approximate a target image. It is a great way of presenting a collection of images that contribute to a common topic. However, existing solutions often require licenses, are time-consuming to use, and produce unsatisfactory results. To address these problems, I developed an open-source desktop application that can efficiently construct photomosaics. Unlike previous works, my solution provides explicit options to trade off between the characteristics of the resulting photomosaic, thus being versatile. Experiments show that my work produces photomosaics with superior quality compared to previous works. My work is available for public use on GitHub[1].

1 Introduction

Images are primary ways to attract audiences. With recent advances in computer vision and image processing, easy-to-use photo-editing applications with collage-making functionalities are becoming increasingly popular. Among these special effects, photomosaic stands out as an engaging way to present a collection of images that contributes to a common topic. Viewed from a distance, a photomosaic is visually identical to a typical photo (target image), while a detailed look will reveal that it is constructed from a grid of smaller images (tiles). A well-known example of photomosaic created as an internet meme is a picture of Leonardo DiCaprio crying, made from the images of Oscar winners.

The quality of a photomosaic is measured by the similarity between the tiles that constitute the mosaic and the target image. Commonly, to enhance the visual quality of the photomosaic, some applications overlay the tiles on top of the target image and make both of them partially transparent. This is known as transparency blending (blending for short). Besides the overall visual similarity to the target image, another important property of a mosaic is the fairness of the tiles. For scenarios such as photomosaic of a graduating class, it is important to make sure that each tile, which is a picture of a graduating student, is used exactly once. There exists a trade-off between the quality of the mosaic and the fairness of the tiles, since fairness requires the frequency of tile use to be considered in addition to the visual similarity. Therefore, it is desirable to design parameters that suitably trade off between the visual quality of the mosaic and the fairness of the tiles depending on applications.

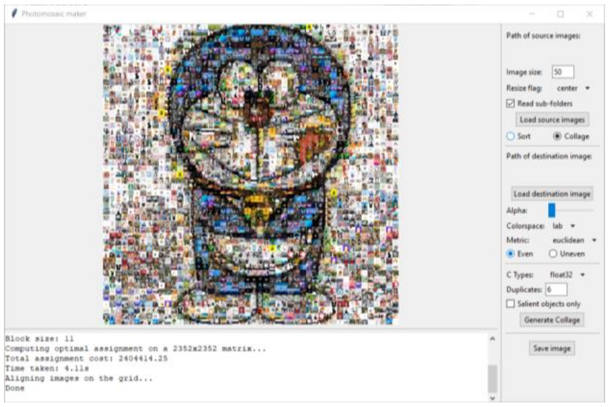


Image size: 50
Resize flag: center
[✓ Read sub-folders]
Sort [✓ Collage]
Alpha:
Colorspace: lab
Metric: euclidean
[✓ Even] Uneven
C Types: float32
Duplicates: 6
Salient objects only

```
Block size: 11
Computing optimal assignment on a 2352x2352 matrix...
Total assignment cost: 2404414.25
Time taken: 4.11s
Aligning images on the grid...
Done
```

Figure 1. User interface of my photomosaic maker

2 Background and Related Works

In the past, photomosaics were commonly crafted by artists and photography specialists by hand. Due to the need to arrange hundreds or thousands of tiles, it is a difficult and tedious job. Although scholar works on photomosaic-building is rare, recent interest in the problem and development of image processing software has led to a few interesting studies. Blasi [1] proposed an efficient method to search for best-match tiles among a large database of tiles, based on the Antipole Tree data structure. Blasi [2] went further with the QuadTree data structure and developed novel ways to render photomosaics. The main problem with these two works is that they cannot be modified to provide options to enforce fairness. Lee [3] proposed a multi-step photomosaic-rendering algorithm based on block matching and color adjustment. While some local fairness and redundancy of tiles are a consideration in Lee's work, no measures are taken to ensure global fairness or strict fair usage of tiles.

Besides scholar works, numerous applications that can build photomosaics can be found online [4, 5, 6]. However, they have one or more of the following shortcomings:

- Require purchased licenses to have full access to all the features (e.g. to download high resolution photomosaic)
- Heavily or solely use blending. Photomosaics should look similar to the target image without blending, but some websites only achieve such similarity with blending. Blending sacrifices the visibility of the tiles. Ideally, users should be able to explicitly set a desired level of blending.

[1] https://github.com/hanzhi713/image-collage-maker

- Users need to upload potentially thousands of tiles before a mosaic can be made, which takes considerable amount of time and has potential privacy concerns.
- Characteristics of the resulting photomosaics cannot be tuned. For example, there is a trade-off between the quality of the photomosaic and the fairness of the tiles (i.e. whether the frequencies of the tiles in the photomosaics are roughly the same).
- The resulting photomosaic looks unsatisfactory or dissimilar to the target image despite the large number of tiles used.

3 Contribution of this Work

My work addresses limitations of existing works. My solution allows users to explicitly set the desired level of blending to trade off between the degree of approximation to the target image and alternation of the color of the tiles. I designed several different algorithms that trade off between the quality of the mosaic and fairness of the tiles. Experiments show that photomosaics built by my algorithms have superior visual quality compared to existing solutions. They are implemented in Python and built on top of efficient math and image processing libraries such as NumPy and OpenCV, ensuring construction of large photomosaics in seconds.

Finally, besides a command-line interface, I also designed a graphical user interface (GUI) for good usability. The current GUI design is shown in Figure 1. The source code is licensed under a permissive open-source license, MIT License, and is available for public use on GitHub. Prebuilt binaries are available for all major platforms.

4 Mosaic Building as An Optimization Problem

4.1 Problem Overview

The objective of photomosaic-building is to find the arrangement of the tiles that best resembles the target image. If the similarity between a grid of tiles and an image can be measured quantitatively by a metric, then the photomosaic-building process can be defined as an optimization problem whose objective is to maximize such a metric. Thus, the performance of the mosaic maker depends on whether the metric can effectively capture the visual quality of the resulting mosaic and whether the optimization algorithm is able to find a global maximum.

Moreover, depending on user preferences, the objective may include the fairness of the tiles. That is, the differences between the frequencies of tiles in the resulting photomosaic need to minimized. This leads to a multi-objective optimization problem which cannot be solved directly, so hyperparameters must be introduced to trade off between the objectives.

4.2 Defining the Metric

While there exist sophisticated algorithms such as feature based methods to determine the visual similarity between two images, the computational simplicity of color distances stand out as we need to evaluate potentially thousands of tiles. Therefore, the color distance is used to determine the quality of a photomosaic. The color distance, referred as cldist for short, between a pair of equal-sized images A and B is given by the vector distance between their pixels

$$\text{cldist}(A,B) = dist\big(flatten(A), flatten(B)\big) \qquad (3)$$

where $dist(\cdot,\cdot)$ is a distance function in $\mathbb{R}^n$ and $flatten(\cdot)$ flattens images in $\mathbb{R}^{w\times h\times c}$ to vectors in $\mathbb{R}^n$ where $n = w \times h \times c$.

There are a number of distance functions between two vectors and a number of difference color spaces and the images can be represented in. For example, Manhattan distance and Euclidean distances can be used as the distance measures. Options for color spaces considered in this work include Blue-Green-Red (BGR), Hue-Saturation-Value(Lightness) (HSV(L)), and CIELAB (LAB). HSV(L) color spaces are often used by artists to adjust the color of an image. BGR is the color space used by digital displays to display the colors to humans. LAB is a less well-known color space, but it has a special property: a uniform change in the color coordinate will lead to a uniform change in the perceived color [7]. The exact choice of the distance metric and color space is left to the users because they will produce different visual effects.

5 Solving the Optimization Problem

Since the multi-objective optimization problem cannot be solved directly, I separated it into two cases and developed specialized algorithms to solve each case. The first case is fair assignment case in which each tile must be used the same number of times. The other case is unfair assignment in which the user can tune a parameter to adjust the redundancy of certain tiles in the mosaic. For both cases, the general strategy is to first divide the target image into equal-sized chunks, and also resize each tile to the size of these chunks. Then, for each chunk, an appropriate tile is selected and matched to this chunk based on Equation 3. Finally, these tiles are assembled to form a photomosaic.

5.1 Notation

Define T to be the set of tiles that will be arrange in a grid to form a photomosaic, and let $N = |T|$ to be the number of tiles in total. Define w_g to be the width of the grid in tiles and h_g to be the height of the grid in tiles. Define w_i and h_i to be the width and height of the target image in pixels. Define B to be the set of chunks that the target image will be divided into, and let the size of each chunk be $k \times k$.

5.2 Solving Fair Assignment

The reason fair assignment is separated as a special case is that it reduces to a well-known problem in combinatorial optimization—linear sum assignment (LAP). LAP arises in many practical scenarios. For example, if there are N workers and N tasks, and there is a cost C_{ij} for worker i to complete task j, and the goal is to find an optimal assignment from workers to tasks such that the total cost is minimized. Similarly, to build a photomosaic given N tiles, we can divide the target image into N equal-sized chunks with side length k and solve LAP with the cost defined as the color distance.

Since LAP requires the number of tiles and chunks to be the same, we need to find out how to divide the target image into chunks. We do so by first determining the grid size to arrange the tiles, and then map the grid to the chunks of the target image. The aspect ratio of the grid $\frac{w_g}{h_g}$ should be very close to the aspect ratio $r = \frac{w_i}{g_i}$ of the target image. Additionally, $N - w_g \times h_g$ should be minimized, meaning that the number of images discarded should be minimized. Since this is a multi-objective optimization problem, I choose to prioritize the minimization of the number of discarded images, as I believe a slight deviation from the true aspect ratio is not noticeable. Therefore, the optimal grid dimension $w_g^* \times h_g^*$ can be calculated as

$$w_g^* = \operatorname*{argmin}_{w_g} \left(\frac{w_g}{\left\lfloor \frac{N}{w_g} \right\rfloor} - r \right), h_g^* = \left\lfloor \frac{N}{w_g^*} \right\rfloor \quad (2)$$

Note that if the user wishes to increase the grid size, they can set a duplication number d, so that each tile used be used d times rather than only 1 time.

After the grid size is obtained, the extra tiles ($N - w_g^* \times h_g^*$) that cannot be put on the grid are discarded. Then, the chunk size is calculated as

$$k = \min \left(\left\lfloor \frac{w_i}{w_g^*} \right\rfloor, \left\lfloor \frac{h_i}{h_g^*} \right\rfloor \right) \quad (3)$$

Then, each tile is resized to $k \times k$ and the target image is resized to $kw_g^* \times kh_g^*$. Now, a one-to-one correspondence between the tiles and the chunks of the target image can be established. Therefore, the cost matrix C can be computed as the pairwise color distances between the tiles and the chunks of the target image.

$$C_{ij} = \text{cldist}(T_i, B_j) \quad (4)$$

Given the cost matrix, the solution of LAP is a bijection between the tiles and the chunks, which is subsequently used to construct the photomosaic.

5.3 Solving Unfair Assignment

For the unfair assignment case, the size of the grid cannot be inferred from the number of tiles, as the frequency of each tile in the resulting photomosaic is unknown. Thus, we require users to specify the grid width w_g^* so that the grid height can be inferred from the size of the target image

$$h_g^* = \text{round}\left(h_i \frac{w_g^*}{w_i} \right) \quad (5)$$

Then, the chunk size k is calculated using Equation (3). Each tile is resized to $k \times k$ and the target image is resized to $kw_g^* \times kh_g^*$.

The most unfair assignment case is the best-match assignment without fairness constraints. That is, each chunk of the target image is assigned with the tile that best matches the chunk, without considering how many times the tile is already used. Therefore, the assignment from tile to chunk can be formulated as

$$B_j \leftarrow \operatorname*{argmin}_{T_i} \left(\text{cldist}(T_i, B_j) \right) \quad (6)$$

To take fairness into account, Equation 6 can be modified to consider the frequency of each tile already used in addition to color distance. Define F_i to be the frequency of the tile T_i currently in the photomosaic. Now, instead of finding the best tile by minimizing the color distances, we minimize the weighted sum of the rank of the tile and the frequency of the tile multiplied by a hyperparameter λ

$$B_j \leftarrow \operatorname*{argmin}_{T_i} \left(\text{rank}(T_i, B_j) + \lambda F_i \right) \quad (7)$$

where $\text{rank}(T_i, B_j)$ is the rank of $\text{cldist}(T_i, B_j)$ among all $\text{cldist}(t, B_j)$ where $t \in T$. Formally, it is given by

$$\text{rank}(T_i, B_j) = \left| \left\{ \text{cldist}(t, B_j) < \text{cldist}(T_i, B_j) \mid t \in T \right\} \right| \quad (8)$$

The hyperparameter λ is a positive real number that trade-off between the quality of the mosaic and the fairness of the tiles. The larger the λ, more weight will be put on fairness. Note that when λ is zero, Equation 7 is identical to 6, which means no fairness is taken into account.

6 Experiments

6.1 Implementation

I chose Python as the language to implement the photomosaic maker, as there is a plethora of high-quality open-source libraries for Python. In the implementation, NumPy and SciPy are used to manipulate matrix and vectors, and OpenCV is used to process and manipulate images. The LAP problem in section 5.2 is solved using the Jonker-Volgenant algorithm [8], implemented in a specialized library by src-d [9]. All these libraries are written in C++, hence ensuring high computational performance, and they provide convenient Python wrappers for easy usage in Python.

6.2 Experiment Setup

The following experiments are conducted to demonstrate the effectiveness of my work:

1. Show the trade-off between quality and fairness when photomosaics are built with different parameters.
2. Compare the photomosaic built by existing solutions against mine.
3. Compare the computation time of existing solutions against mine.

Unfortunately, there is no well-defined quantitative metric for the quality of photomosaics, and therefore the comparison is limited to qualitative (visual) examination.

The tiles used in the experiment are collected from the publicly available profile pictures of my friends. The target image is a picture of Doraemon. They are illustrated in Figure 2.

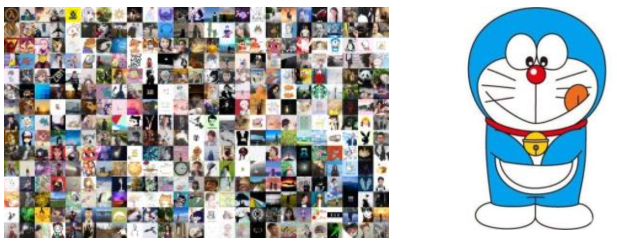

(a) Tiles (b) Target Image

Figure 2. Tiles and target image for the experiment

[1] https://github.com/hanzhi713/image-collage-maker

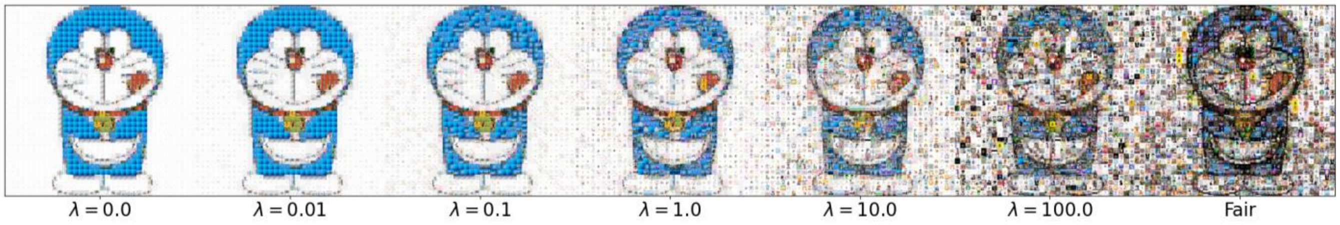


Figure 3. Trade-off between quality and fairness. Color space is LAB. Grid size: 36x36

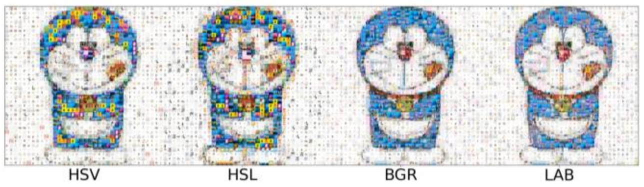
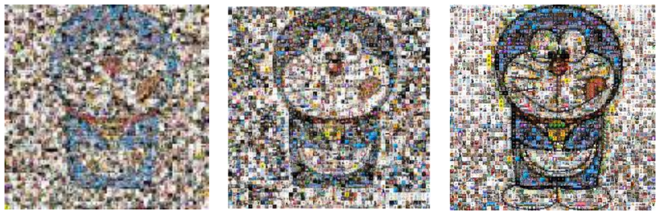


Figure 4. Effect of different color spaces. Grid size: 36x36. $\lambda = 1$



(a) mosaically.com (b) easymoza.com (c) My Result

Figure 5 Compare my result with othes. Grid size: 50x50

6.3 Results

Figure 4 show the photomosaic built when the target image and the tiles are represented in different color spaces. For this particular example, the BGR and LAB color space give better visual result. However, this conclusion may not hold true for other target images, and thus I leave the color space as an option (default to LAB) chosen by the user.

Figure 3 shows the photomosaic built with different values of λ for the unfair assignment case. The result from the fair assignment case is also there for reference. It can be clearly observed when λ increases, a more diverse collection of tiles will be used to construct the photomosaic, but the overall visual quality slowly diminishes. $\lambda = 1$ seems to be a reasonable compromise between quality and fairness. One important note is that when λ is large and the tile assignment is close to fair assignment, it is better to use the fair assignment solver instead, because it can guarantee a global optimal solution while enforcing strict fair tile usage. This effect can be seen when comparing $fair$ and $\lambda = 100.0$ in Figure 3. Both photomosaics have similar fairness, but the $fair$ mosaic has better visual quality compared to the $\lambda = 100.0$ mosaic.

Figure 5 compares the photomosaic built by my algorithm and two commercial photomosaic building websites. Since they require payment to download high resolution result, the figures are simply screenshots. The reason to select these websites is that they are the few that can have blending disabled and support fair assignment. For fair comparison, all photomosaics are generated with the same grid size. All mosaics are generated under the strict fairness constraint. For my result, the LAP color space is used. It can be clearly seen from the figures that my result better approximates Doraemon and has better visual quality.

Finally, to illustrate the computational efficiency of my implementation, I recorded the time taken for the commercial websites and my implementation to produce the photomosaics in Figure 5. The result is listed in Table 1. It can be seen that my solution is much faster than theirs. Note that the time required to upload the tile to the websites (few minutes) and the time required to read the images from disk for my implementation (a few seconds) are not included in the table. Therefore, in practice, my solution will lead to significantly less waiting time for the user, thus allowing the user to experiment with more combinations of parameters to achieve the desired effect.

Table 1 Computation Time of Photomosaics in Figure 5

	mosaically.com	easymoza.com	My solution
Time (s)	14	45	4

7 Conclusion

My open-source versatile photomosaic maker is designed to consider fairness of tiles. I use specialized algorithms to solve different cases depending on the level of tile fairness desired by the user. My photomosaic maker is built on top of highly-efficient computing libraries to ensure fast photomosaic construction, but it hides all the complexity behind a user-friendly interface. Experiment results show that my work surpasses the performance of a number of commercial photomosaic-making websites in terms of both speed and quality.

8 Future Work

A number of extensions of the basic photomosaic generation algorithm can be explored. For example, one can relax the restriction that each tile needs to be a square image or apply the algorithm to videos. Moreover, some tricks to enhance the overall effect such as color adjustment proposed by Lee [3] can be implemented.

REFERENCES

[1] Blasi, G.D., & Petralia, M.P. (2005). Fast Photomosaic.

[2] Blasi, G. D., Gallo, G., & Petralia, M. P. (2006). Smart Ideas for Photomosaic Rendering. In S. Battiato, G. Gallo, & F. Stanco (Reds), 4th Eurographics Italian Chapter Conference. doi:10.2312/LocalChapterEvents/ItalianChapConf2006/267-271

[3] Lee, H. (2014). 'Generation of Photo-Mosaic Images through Block Matching and Color Adjustment'. World Academy of Science, Engineering and Technology, Open Science Index 87, International Journal of Computer and Information Engineering, 8(3), 457 - 460.

[4] EasyMoza lets you create a photo mosaic online. (n.d.). Retrieved November 1, 2021, from https://www.easymoza.com/

[5] Mosaically - Free Online Photo Mosaic Creator. (n.d.). Retrieved November 1, 2021, from https://mosaically.com/

[6] Free online photo collage, photo grid, and photo mosaic maker. (n.d.). Retrieved November 1, 2021, from https://www.picmyna.com/

[7] Luo, M. R. (2016). CIELAB. In M. R. Luo (Red), Encyclopedia of Color Science and Technology (bll 207–212). doi:10.1007/978-1-4419-8071-7_11

[8] Jonker, R., & Volgenant, A. (2005). A shortest augmenting path algorithm for dense and sparse linear assignment problems. Computing, 38, 325-340.

[9] src-d. (2021). Linear Assignmment Problem solver using Jonker-Volgenant algorithm - Python 3 native module. GitHub repository. Retrieved from https://github.com/src-d/lapjv

[1] https://github.com/hanzhi713/image-collage-maker