

Projet de Reconnaissance Visuelle Assemblage Automatique d'un puzzle

Dylan Sechet, Xavier Jeunot, Agathe Gioan

April 17, 2023

Abstract

Notre article présente le développement d'un programme de vision par ordinateur pour l'assemblage automatique d'un puzzle. Les puzzles classiques sont un divertissement à portée éducative car ils entraînent le cerveau humain à la reconnaissance de schémas, améliorent la coordination et développent la conscience spatiale [1].

Notre programme a été créé pour assembler automatiquement les puzzles en utilisant une image des pièces mélangées comme entrée, sans référence à l'image originale du puzzle. Notre solveur de puzzle a été développé en utilisant plusieurs concepts de vision par ordinateur et des travaux antérieurs dans le domaine. L'idée de notre algorithme est de détecter les pièces et leurs contours, et de les assembler en utilisant les formes de chaque pièce. Nous avons testé avec succès notre solution sur un puzzle de 12 pièces.

Cette solution automatique d'assemblage de puzzle pourrait constituer un travail préliminaire à des applications beaucoup plus concrètes, comme par exemple le réassemblage de documents endommagés.

1 Introduction

Le but de ce projet est de créer un programme de résolution de puzzle qui assemble les pièces à l'aide d'une image des pièces démontées, sans utiliser une image de référence du puzzle assemblé. Le résolveur a été implémenté en python et utilise largement les outils de la bibliothèque OpenCV . Pour réaliser ce projet, nous nous sommes notamment inspirés des travaux d'un projet d'étudiants de Stanford [1] et d'un blog scientifique qui traitait de ce sujet [2]. Nous avons cherché des images de puzzle pour pouvoir construire et tester notre modèle. Les images recherchées devaient respecter certains critères pour nous permettre d'appliquer notre algorithme :

- Les pièces de l'image initiale ne doivent pas se chevaucher, ni se toucher.
- L'image doit être capturée sans distorsion due à un mauvais angle de la photo.
- Les pièces sur l'image source constituent bien un puzzle, il n'en manque aucune, aucune autre pièce provenant d'un autre puzzle ne sont insérées.

- Il s'agit de pièces d'un puzzle classique, avec des formes standards, caractérisées par quatre coins, des têtes et des creux qui permettent d'emboîter deux pièces, et de bords plats pour les pièces constituant les bords du puzzle.
- Nous avons fait le choix de travailler avec des images au fond uni pour se concentrer sur la résolution du puzzle plutôt que sur la détection des pièces dans un contexte bruyant.

Nous avons trouvé plusieurs images correspondants à nos critères et avons sélectionné une image de puzzle illustré Figure 1.

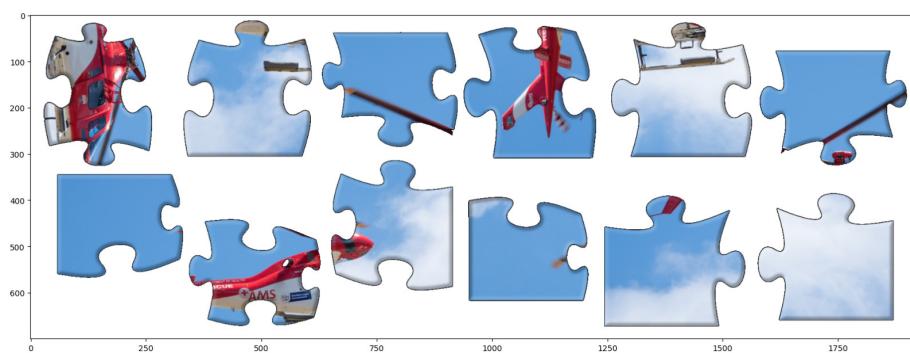


Figure 1: Image source des pièces du puzzle à reconstituer

Après avoir sélectionné l'image de puzzle, nous avons traité le projet et imaginé une solution de la façon suivante :

1. *Détection des pièces* : La première étape consiste à détecter grossièrement les pièces présentes sur l'image et à les transposer sur un fond uni.
2. *Détection des contours* : Les contours des pièces sont ensuite précisément détectés pour pouvoir utiliser leur forme lors de l'assemblage.
3. *Détection des formes* : Nous identifions les différentes formes constituant les contours de chaque pièce (les coins, les têtes, les creux et les bords plats).
4. *Assemblage* : Nous assemblons les pièces grâce à un algorithme de matching s'appuyant sur l'analyse des formes des contours des pièces. Pour cela, nous détectons la forme constituante chaque bord (plat, tête ou creux), et nous comparons tous les bords potentiellement compatibles grâce à une fonction de similarités que nous détaillons plus bas.

Dans les sections suivantes, nous décrirons les travaux existants sur le sujet, ceux dont nous nous sommes inspirés et les autres méthodes existantes, puis nous décrirons en détails les quatre étapes de notre méthode citées plus haut. Enfin, nous présenterons les résultats obtenus sur notre exemple.

2 Travaux semblables

La résolution automatique de puzzles est un problème classique de vision par ordinateur qui a été traité de nombreuses fois dans la littérature. H. Freeman et L. Gardner sont les premiers à s'y être intéressés en 1964 [3]. Leur méthode de résolution s'appuie uniquement sur la forme des pièces, et l'algorithme développé est capable de résoudre par exemple des puzzles blancs. Depuis, de nombreux progrès ont été réalisés et de nouvelles méthodes de matching ont été développées. En 2013, Sholomon *et al.* ont dévoilé un algorithme génétique capable de résoudre des puzzles de plus de 22 000 pièces dans un temps raisonnable [4]. En 2016, Travis V. Allen développe un programme qui prend en compte la forme des bords mais également la couleur [1]. Cette méthode permet de rendre le matching plus précis et plus robuste en considérant deux critères indépendants. Travis V. Allen décrit par ailleurs dans ce papier une méthode que nous avons testée et qui consiste à représenter les contours de chaque pièce en utilisant des coordonnées polaires et qui permet caractériser la forme des pièces [1].

3 Méthode

3.1 Détection des pièces

La première étape dans la résolution du puzzle est l'identification de chacune des pièces. Nous voulons réussir à détecter grossièrement les pièces pour les séparer du fond de notre image. Nous avons commencé par utiliser une opération de seuillage suivie de la détection des composantes connexes pour obtenir le nombre de pièces de puzzle sur l'image de base, ainsi que d'obtenir des masques pour chaque pièce. Dans notre cadre, le fond uni permet de se dispenser de filtrage, et nous devrions sans doute dans des conditions moins contrôlées appliquer un filtrage sur la surface des composantes connexes obtenues pour éliminer ceux dûs au bruit de fond. L'opération de seuillage permet finalement d'obtenir un label spécifique correspond aux pixels de fond qui n'appartiennent à aucune pièce. Ainsi, nous pouvons identifier les pixels appartenant à des pièces de ceux appartenant au fond et détecter les pièces présentes sur l'image comme le montre la Figure 2.

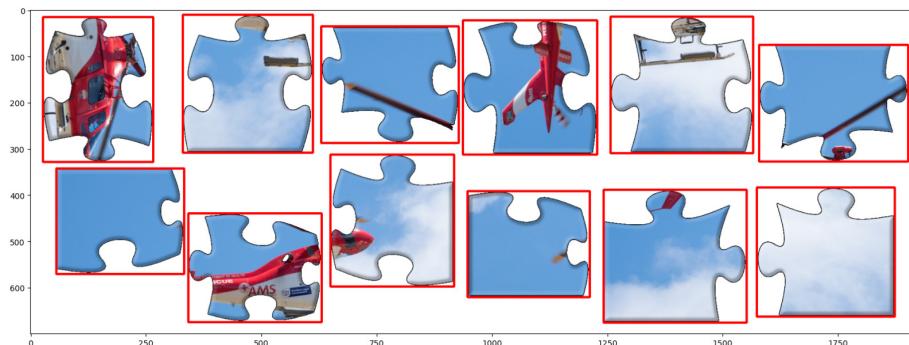


Figure 2: Identification des pièces de puzzle sur l'image

3.2 Détection des contours

Notre algorithme de matching s'appuie sur l'analyse des formes des contours des pièces. Cela nécessite donc d'obtenir pour chaque pièce une liste correspondant aux pixels qui constituent son contour. L'intérieur des pièces n'est en effet pas utilisé dans notre algorithme de matching et ce sont seulement les bords qui ont un sens pour notre problème. Les contours sont aisément obtenus à partir des composantes connexes détectées à l'étape précédente, et peuvent être observés dans la Figure 3.

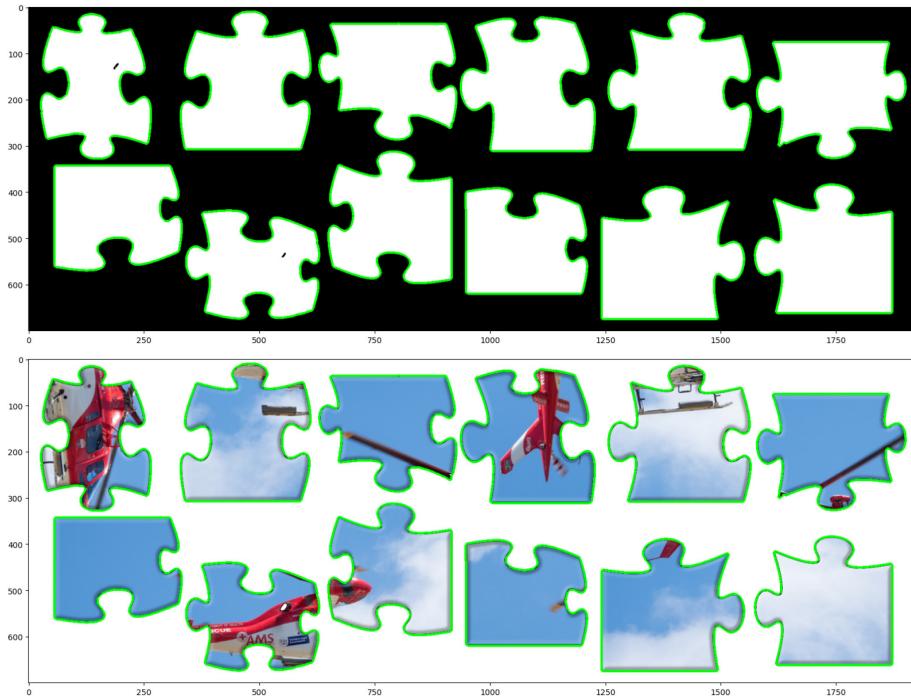


Figure 3: Identification des contours de chaque pièce

3.3 Détection des formes

Notre algorithme repose sur la mise en correspondance bord à bord des pièces. Cela demande non seulement d'identifier les 4 côtés de chaque pièce, mais aussi de les catégoriser comme tête, creux, ou côté plat.

Nous avons d'abord essayé de reprendre le travail de T. V. Allen [1] et de commencer par détecter les coins de chaque pièce grâce à leurs coordonnées polaires. Pour cela, nous sommes partis des coordonnées xy de chaque pixel constituant chaque bord de chaque pièce, que nous avons convertis en coordonnées polaires par rapport au centreïde de chaque pièce. Cette transformation permet notamment d'identifier les quatre coins de chaque pièce. Pour ce faire, on étudie le graphe (θ), dans lequel les pics correspondent à des changements d'angles dans le contour. Cependant, les pics correspondant aux têtes et creux apparaissent

également, ainsi qu'une certaine quantité de bruit. Nous avons tout d'abord filtré la majorité du bruit en éliminant tous les pics au sommet inférieur à la médiane du signal, et en éliminant ceux ayant une largeur ou un proéminence trop faible. Après avoir constaté que les pics associés aux coins de la pièce avaient une forme plus triangulaire que celles liées aux creux et trous, nous avons utilisé le rapport entre la largeur du pic à la moitié de sa hauteur et sa largeur à 10% de son sommet pour caractériser les pics "triangulaires". Sur la figure 4, on observe en vert ces pics triangulaires correspondant aux coins de la pièce, et nous conservons les quatre pics qui ont le plus grand rapport longueur rouge sur longueur verte.

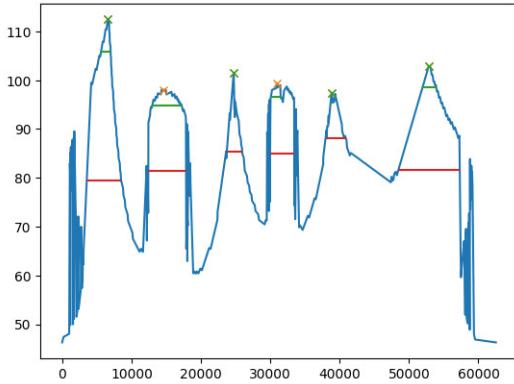


Figure 4: Détection des coins grâce aux coordonnées polaires, les croix vertes correspondant à des coins

Nous pensions pouvoir détecter les quatre coins de chaque pièce en jouant sur les hyperparamètres de détection des pics. Malheureusement, la détection des contours s'est révélée trop peu précise à cause du bruit, et la détection des pics était extrêmement sensible aux hyperparamètres et manquait singulièrement de robustesse. Nous avons donc décidé de nous rabattre sur une autre technique pour détecter les formes de chaque pièce.

Nous nous sommes alors inspirés de la méthode utilisée par des ingénieurs d'AbtoSoftware [2]. La méthode consiste désormais à identifier les creux et les têtes comme dans la figure 6. Il devient alors possible de remplir ces derniers pour obtenir une pièce quasi-rectangulaire, et dans laquelle détecter les coins est bien plus facile.

Pour réussir à identifier ces creux et têtes, nous commençons par tracer l'enveloppe convexe de chaque pièce. Cela nous permet d'identifier tous les points défauts de convexité du contour de la pièce. Ceux suffisamment éloignés de l'enveloppe correspondent aux débuts des creux et des têtes des bords, comme le montre la figure 5.

Une fois ces points détectés, il faut déterminer s'il s'agit d'une paire de points associée à une tête, ou à un point unique associé à un creux. Pour ce faire, pour chaque paire de points successive, on essaie de déterminer s'il contient une tête. On utilise pour cela le fait que les têtes sont quasi-circulaires en définissant la circularité d'un polygone de périmètre P et de surface S par: $C = \frac{S}{P^2}$. Pour

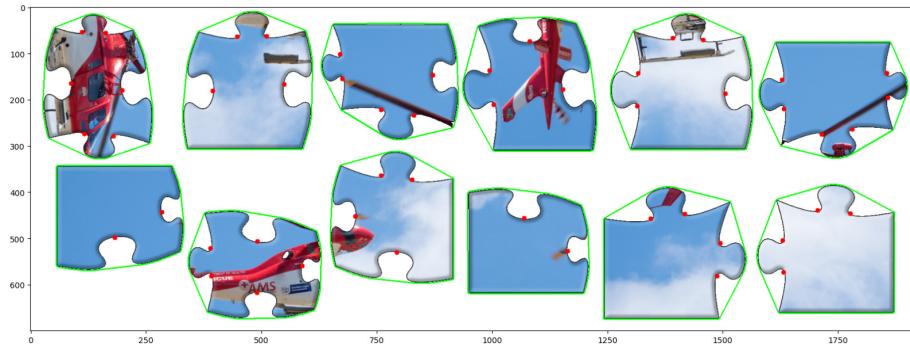


Figure 5: Détection des points formant les creux et les têtes

une cercle, on a $C_{\text{cercle}} = \frac{\pi r^2}{(2\pi r)^2} = \frac{1}{4\pi}$. Si la circularité du contour entre deux défauts de convexité est proche de cette valeur, on considère qu'il s'agit d'une tête, et on élimine ces deux points. Une vérification de l'aire du contour est aussi faite pour éliminer les quelques faux positifs : si elle est supérieure à 30% de celle de la pièce, les points ne sont pas considérés comme des têtes. Lorsque toutes les paires de points ont été parcourus, ceux qui demeurent correspondent à des creux. Grâce à la détection de ces points, nous pouvons ensuite supprimer les têtes et remplir les creux (Figure 6), et obtenir la figure 7 où les seuls angles restants sont les coins de chaque pièce.

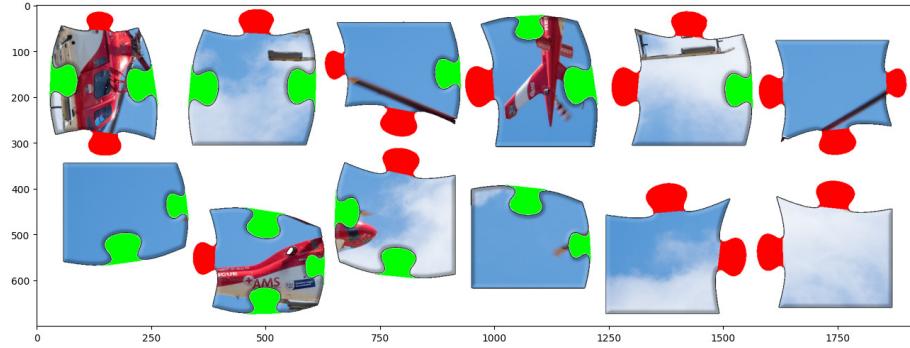


Figure 6: Identification des têtes et creux de chaque pièce

Ce découpage nous permet d'obtenir tous les coins de chaque pièce en les approximant par un quadrilatère et en considérant que les sommets de ce dernier correspondent aux coins. Il devient alors possible pour chaque pièce d'identifier les quatre bords et de les catégoriser comme tête, creux, ou zone plate, comme on peut le voir dans la Figure 8.

3.4 Assemblage

Après toutes ces étapes pour identifier chaque bord et les formes les composant, nous détaillons ici l'étape d'assemblage des pièces. Pour chaque bord présentant

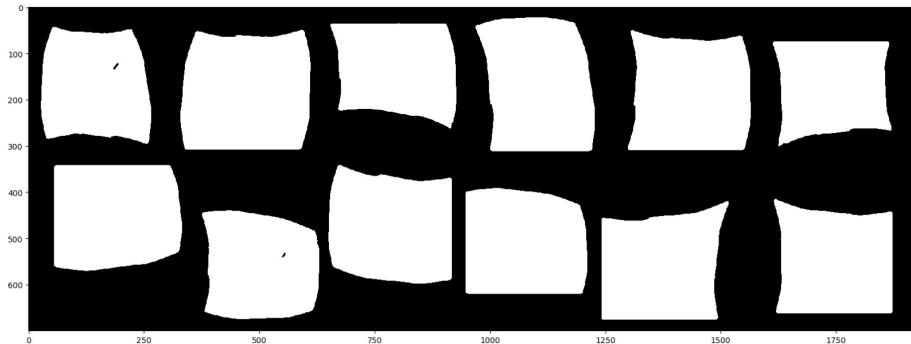


Figure 7: Suppression des creux et des têtes de chaque pièce

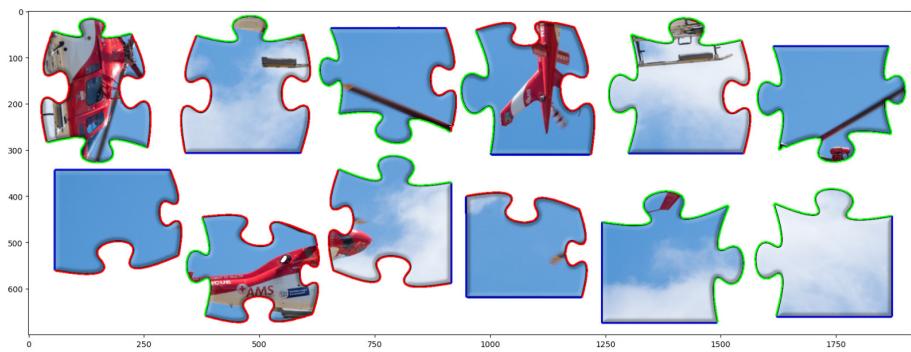


Figure 8: Détection des formes de chaque pièce. Les côtés en bleu sont planes, les côtés rouges des creux et les côtés verts des têtes.

un creux ou une tête, nous allons chercher parmi l'ensemble des bords potentiellement complémentaires (les bords présentant la forme opposée à notre bord), le bord ayant le plus de complémentarité. Pour cela, la première étape est d'aligner chaque bord sur le même axe. Nous nous inspirons pour cela d'une méthode trouvée dans un dépôt git [5]. Nous traitons tous les bords à tête ou à creux en leur appliquant une translation et une rotation pour placer le premier point de chaque bord à l'origine, et le dernier point sur l'axe des ordonnées. Cela permet d'obtenir des bords à peu près verticaux. Les bords ayant subi cette transformation seront dits *normalisés*. Nous introduisons ensuite une fonction de similarité entre deux bords. Cette fonction permet pour chaque bord d'obtenir le bord ayant le plus de complémentarité, c'est à dire celui ayant la distance plus petite au bord en question. La distance entre un bord a et un bord b est défini comme suit : pour chaque point x du premier bord a normalisé, on calcule la distance de x au point le plus proche du bord b normalisé. On somme ces distances pour chaque point x du bord normalisé a pour obtenir la similarité entre ces deux bords. Les bords normalisés étant à peu près verticaux, l'idée est que les bords complémentaires se chevauchent presque, présentant une distance totale très faible.

Nous commençons par assembler le contour extérieur de notre puzzle : nous détectons un premier coin (une pièce à deux bords plats) et associons, grâce

à la méthode expliquée précédemment, son voisin de droite à bord plat, et nous répétons l'opération jusqu'à avoir le contour complet. Nous passons ensuite à l'intérieur du puzzle en commençant par matcher les bords internes des pièces que nous avons déjà placé sur les bords. Nous réussissons ainsi à placer itérativement toutes les pièces internes du puzzle.

4 Résultats

Nous avons donc testé notre algorithme sur un puzzle de 12 pièces représentant une photographie d'un hélicoptère. Pour ce puzzle, l'algorithme:

- détecte correctement les contours de toutes les pièces,
- caractérise correctement la forme (creux, tête ou plat) des 48 bords,
- identifie correctement les coins de toutes les pièces,
- positionne correctement toutes les pièces.

Nous avons d'abord testé notre algorithme sur des pièces du puzzle déjà bien orientées, puis nous avons fait d'autres tests en appliquant des rotations aléatoires à nos pièces, en réussissant à chaque fois à reconstituer le puzzle correctement. Le puzzle assemblé est présenté Figure 9.



Figure 9: Résolution du puzzle hélicoptère par notre algorithme

Nous avons voulu tester notre algorithme sur un puzzle plus complexe, trouvé sur un dépôt git traitant du même sujet [6] contenant 30 pièces et représentant "La nuit étoilée" de Van Gogh (Figure 10).

Nous avons réussi les étapes de détection des pièces, des contours et des angles. Malheureusement, notre fonction de similarités n'était pas assez robuste pour bien réaliser la partie assemblage : nous avons pu reconstituer le premier bord du puzzle, mais la fonction d'assemblage n'a pas réussi à correctement détecter tous les contours.

Pour évaluer plus précisément notre algorithme, il conviendrait de le tester sur des puzzles avec différents nombres de pièces, avec des qualités d'image différentes et avec des formes de pièces plus ou moins marquées.



Figure 10: "La nuit étoilée" de Van Gogh, puzzle de 30 pièces

5 Conclusions

Nous avons entièrement créé un résolveur de puzzle automatique sur Python en utilisant les outils de la librairie OpenCV. Nous avons réussi à reconstituer notre puzzle en utilisant uniquement la forme des pièces du puzzle.

Nous avons identifié certains points qui pourraient permettre de rendre notre algorithme de résolution automatique plus robuste et plus performant, notamment pour traiter des puzzles avec davantage de pièces et des images moins nettes que celle utilisée dans ce papier. D'abord, nous avons eu des problèmes sur l'utilisation des coordonnées polaires pour caractériser les formes de nos pièces alors que cette méthode avait fait ses preuves dans des travaux semblables [1]. Nous comprenons que les contours que nous avons obtenus étaient trop bruités et qu'un travail supplémentaire sur l'élimination de ce bruit permettrait d'être plus performant sur l'identification des caractéristiques géométriques des pièces. En effet, lors de l'identification des contours, notre algorithme extrapole les bords pour identifier les pièces, mais nous ne détectons en réalité que quelques points de chaque bord à cause du bruit et du fond. Avec plus de travail sur cette détection, nous pourrions avoir des contours plus précis et notre fonction de similarités seraient plus performante. Nous aurions également souhaité pouvoir intégrer dans l'algorithme de matching la prise en compte des couleurs des contours des pièces de façon à avoir un matching plus précis et plus robuste. Nous voyons dans ces points des pistes d'amélioration de notre algorithme pour un travail futur.

References

- [1] Travis V. Allen. Using Computer Vision to Solve Jigsaw Puzzles. 2016.
- [2] Computer Vision Powers Automatic Jigsaw Puzzle Solver, April 2019.
- [3] H. Freeman and L. Garder. Apictorial Jigsaw Puzzles: The Computer Solution of a Problem in Pattern Recognition. *IEEE Transactions on Electronic Computers*, EC-13(2):118–127, April 1964.
- [4] Jordan Davidson. A Genetic Algorithm-Based Solver for Very Large Jigsaw Puzzles: Final Report. 2015.

- [5] Ken Ellinwood. PuzzleSolver 2019, April 2023. original-date: 2019-01-24T20:34:12Z.
- [6] Julian Minder. jigsaw-puzzle-solver, October 2022. original-date: 2020-01-17T20:45:06Z.