UWAGA: wymyslic prawidlowa nazwe pracy + poprawic zlozonosci + opisac dwa algorytmy

Lord Vorotron: Finding the Best JFA Variant for the Coming Winter

MACIEJ A. CZYZEWSKI, Poznan University of Technology, Poland
KAMIL PIECHOWIAK, Poznan University of Technology, Poland

This paper studies a practical usage of machine learning (AutoML) to automate research towards discovering efficient Voronoi Diagram and Distance Transform algorithms. As the baseline we used the Jump Flooding Algorithm (JFA) - by finding new mutations which works best for specific data, and then ensembling them into one, we create new state-of-the-art algorithm in this field named **Vorotron** with time complexity ≈O(1) and work complexity ≈O(N). The algorithm is faster and produces more accurate approximations. It could be extended into 3D space in a slice-by-slice manner. We started from the assumption that JFA has potential for improvement - some benefits can be observed for specific data by adding random noise and adjusting the step size in JFA. This showed us that, AutoML could examine this space, and find the best possible algorithm in each case. In the further part of the work, we discuss the results, compare the variants and ensemble for creating the final algorithm.

CCS Concepts: • **Computing methodologies** → **Computer graphics**; **Parallel computing methodologies**; • **Theory of computation** → *Randomness, geometry and discrete structures*; **Data structures design and analysis**.

Additional Key Words and Phrases: Voronoi Diagram, Distance Transform, Code Generation

1 INTRODUCTION

This paper[1] studies a practical usage of machine learning to automate research towards discovering efficient Distance Transform algorithms (utilizing technique known as AutoML). Thus, by finding mutations which works best for specific data, and then ensembling them into one, we create new state-of-the-art algorithm in this field named Vorotron with time complexity ≈O(1) and work complexity ≈O(N).

Notable contribution to the quick algorithm that makes Distance Transform (DT) using graphics hardware includes [4] that creates a cone for each input (point/seed) and renders those cones to obtain the Voronoi diagram as the lower envelope of these cones. [3] use planes tangent to a paraboloid and thus avoid the errors caused by the tessellation of the cones. Unfortunately, the drawback of this approach is the significant amount of computation and the implementation complexity.

Jump flooding algorithm (JFA)[2] is an interesting way to utilize the graphics processing unit to efficiently compute Voronoi diagrams and distance transforms [7]. This method is faster and produces more accurate results [8], and furthermore, it could be extended into 3D space in a slice-by-slice manner. This is more effective than the previous research carried out by [10], because the speed of JFA is almost independent to the number of seeds [8].

Based on this research and findings, several efficient GPU-based algorithms which are either work optimal or time optimal have been proposed including SKW [9], PBA [1], FastGPU [2], Honda's algorithm [5] and WTO [6].

[1] the original title for this paper was "Lord Vorotron: Finding the Best JFA Variant for the Coming Winter"
[2] a novel pattern of communication

Authors' addresses: Maciej A. Czyzewski, maciejanthonyczyzewski@gmail.com, Institute of Computing Science, Poznan University of Technology, Poznan, Poland; Kamil Piechowiak, kamil.cams@gmail.com, Institute of Computing Science, Poznan University of Technology, Poznan, Poland.

The main question that needs to be addressed now is whether JFA has potential for improvement. We found some benefits for specific data by adding random noise and adjusting the phase size in JFA. Therefore, this shows that, AutoML could examine this unknown space, and find the best possible algorithm in each case.

For convenience, this work focus on the Voronoi diagram only - because this problem can be translated to DT [7]. The algorithm would be an approximation of the output, thus we suggest using WTO [6] for exact DT (EDT). The major contributions of this paper are thus:

(1) Presenting new state-of-the-art variants of algorithm for Voronoi Diagram and Distance Transform:
JFAStar - (Multi-domain) single best variant; **Vorotron** - (Specific-domain) ensemble of weak variants; and
(2) Analyzing all possible variants of JFA: comparing error and speedup relative to bruteforce method

2 RELATED WORK

Several efficient GPU-based algorithms which are either work optimal or time optimal have been proposed including JFA [7], SKW [9], PBA [1], FastGPU [2], Honda's algorithm [5] and WTO [6].

Reference	Algorithm	Exactness	Time	Work
[2]	FastGPU	Exact	$O(n^3/p)$	-
[1]	PBA	Exact	$O(n)$	$O(mN)$
[5]	based on SKW	Exact	$O(n)$	$O(N)$
[6]	WTO	**Exact**	$\mathbf{O(\log n)}$	$O(N)$
[9]	SKW	Approximate	$O(n)$	$O(N)$
[7]	JFA	Approximate	$O(\log n)$	$O(N \log n)$
In this paper	JFAStar	**Approximate**	$\mathbf{\sim O(\log^* n)}$	$\sim O(N)$
	Vorotron	**Approximate**	$\mathbf{\sim O(1)}$	$\sim O(N)$

Table 1. Different GPU algorithms for computing EDT

2.1 Jump Flooding

redukcja i bridge pomiedzy intro (usunac subsection)

co to jest jump flooding? tak naprawde to nie jest algorytm do voronoi-a tylko pattern komunikacyjny w programowaniu rownoleglym - swojej pracy doktorskiej autor tej techniki podaje wiele zastosowan jednak w swoich badaniach ogranicza sie do Voronoi-a. glownym pytaniem roznych takich patternow jest ile potrzebnych jest rund/operacji komunikacji aby zagwarantowac aby dana informacja zostanie dostarczona. akurat w voronoi-u wiele komorek jest lokalna w skali calego przykladu - wiec JFA ktora gwarantuje dostarczenie informacji globalnie do kazdego punktu - wykonuje pewne niepotrzebne operacje. szybkosc i zajetosc pamieciowa JFA jest satysfakcujaca, jednak proste modyfikacje pokazuja ze algorytm ten wykonuje sie szybciej w pewnych przypadkach (i to typowych). dlatego naturalnym pytanie powinno byc w jakich oraz jakie modyfikacje wplywaja na szybkosc dzialania.

2.2 AutoML

przeniesc do Proposed Method

https://arxiv.org/pdf/1801.09373.pdf https://arxiv.org/pdf/1804.10120.pdf https://arxiv.org/pdf/1703.06353.pdf

okay przenioslem - dodac prace co tez tak szuka algosow

3 PROPOSED METHOD

przepisac ten szkic bo jezyk sie placzy

JFA opiera sie na tym ze infomacja jest przekazywana ??????. Przekazanie odbywa sie w log(n) krokach. Wiec przeprowadzilismy krotki eksperyment applyujac losowy szum na wejsciowa masce. Okazalosie sie ze ilosc potrzebnych krokow spadla - powstaly losowe shortcuty. Co oznacza ze powinny istniec inne "mutacje" algorytmow lepsze w pewnych okreslonych przypadkach. Wiec szukanie najszybszego algorytmu bedzie nastepujace:

- Wymyslenie wszystkich mozliwych wariantow JFA
- Mutacje i zapisanie najlepszych wersji dla danej domeny
- Ensemblacja algorytmow tak aby wybierac najlepszy variant dla danej domeny

3.1 Domain Space

okay, zupelnie inaczej tutaj podejsc, opisac jakie sa przypadki i jakie sa spotykane

jakie domeny i dlaczego (i jak dzialaly gen_uniform, gen_polar, gen_grid)

KWADRAT TUTAJ????????? (komentarz latex)

Shape / **Density**	Small (32-128)	Medium (256-448)	Large (512-1536)
Low (ρ=0.00005-0.001)	Bruteforce	Bruteforce	JFA
High (ρ=0.01-0.1)	JFA	JFA	JFA

Table 2. State-of-the-art for specific domains (before our work).

3.2 Search Space

UWAGA: opisac ze szukamy 4 podprzestrzenie w turach! (moze tez o nowym wzorku ktory szuka tylko prawidlowych

bridge z score function gdziekolwiek to bedzie obliczyc ile jest aktulanie wersji algosow np. czy jest to juz 2do14 jak mamy 3xreal w wielomianie AKTUALNIE JEST okolo 7,200?

jakie modyfikacje, na to osobna sekcja? wiec co tu napisac chyba tylko o zlozonosci problemu i ze kod jest skladany i testowany a niektore wersje sa pomijane zgodnie z dzialaniem gp_minimize (Bayesian optimization using Gaussian Processes).

w naszym wypadku zdefiniowalismy pewien zbior wariantow pewnych czesci algorytmu (Search Space), modul testujacy dana mutacje/wariant - sklada kod kernela a pozniej go weryfikuje na naszej Domain Space.

3.3 Score Function

JEST PROBLEM - zdazaja sie warianty ktore 0zeruja rzadkie i sa bardzo szybkie na duzy matrycach - trzeba do ponownie zbalansowac roznica w pikselach pomiedzy bruteforce a algorytmem - napisac o tym / tez ze to wszystko to ilorazy do bruteforce dla voronoi-a interesuja nas 2 parametry Error oraz Szybkosc, aby wyniki byly wiarygodne porownujemy je z bruteforcem (a wiec bedzie to iloraz). aby ocenic dana mutacje musimy przypisac jakis Score danej wersji, wiec uzylismy wzor ponizej

$$S(x, y) = max\{0, \sqrt{x} \cdot (100 - y^2)\}, \tag{1}$$
$$0 \leq y \leq 100, 0 < x \tag{2}$$

ktory kaze za zbyt wysokie errory, dajac zerowy wynik - skladnik przy y rosnie szybciej niz x wiec gdy przekroczy 100 da nam ujemny wynik - czyli 0.

3.4 Optimizer

opisac dwie osobne taktyki optymalizacji dla best single vs. ensemble

mozemy napisac ze korzystalismy z forest/gp minimize, ale tez wspomniec ze aby miec najlepszy best single to trzeba bylo optymalizowac rownoczesnie cala przestrzen (od malych do duzych, gestych po rzadkie), a zeby miec najlepszego Vorotrona - czyli ensembla to trzeba bylo dla kazdej domeny z optymalizowac a pozniej jedynie zrobic balancera!!!!!!!!!!!

trzeba to przeszukiwac tak aby nie sfiksowal na zadanych parametrach - bo niechcacy ocenia na poczatku ze szum/dual jest nie fajny i pozniej go juz nie rozwaza

3.5 Ensemble

mozna zapisac tabele dla kazdej domeny (shape) / i zrobic przewidywania parametrow (slownik wariantu) - bo dla malych oplacalne sa Circle6 a dla wiekszych Circle12 i tak dalej - jak to zrobic?

patrzac na rezultaty mozemy znalesc jaki algorytm najlepiej sprawdza sie w zadanej domenie. np. widac ze dla malej ilosci seedow (malo gestych przypadkow, ktore maja mala powierzchnie) oplaca sie uzyc bruteforce. Dla kolejnych wiekszych przypadkow innych wariantow JFA. Jak wybrac algorytm? Kazdy przypadek ma 'shape' oraz 'num' wiec mozna na CPU wysemplowac pare punktow albo odrazu obliczyc gestosc i wybrac odpowiedni algorytm. To takich ensemblacji najlepiej sprawdzi sie drzewo decyzyjne (moze byc boostowane).

4 VARIANTS

ZROBIC LADNE RYSUNECZKI w Google Slides - eksport to pdf!

To compute the Voronoi diagram for a 2D grid of size n×n with a given set of seeds at some grid points, we are interested to propagate the content (in particular, position information) of each seed s to each grid point so that each grid point can decide which seed is its closest one.

Niektore operacje propagacji informacji sa zbedne - tylko w przypadkach rzadkich macierzy potrzeba jest log(n) krokow aby uzyskac prawidlowy wynik. Rozne warianty omawiane w [8] pozwalaja zredukowac blad klasycznego JFA. Nie zostaly jednak omawiane przypadki gdzie poszczegolne modyfikacje sa uzywane z innymi.

Dlatego w tej pracy prezentujemy dodatkowe modyfikacje ktore mozna zastosowac aby stworzyc nowe warianty. Pewne modyfikacje sa oczywiste i wynikaja z alternatywnego podejscia (zamiast anchoru[3] kwadratowego mozna uzyc kola), informacje mozna wstepnie rozpropagowac losowo - w nadzie ze pozwoli nam skonczyc algorytm w mniejszej ilosci krokow.

Aby badania byly bardziej przejrzyste trzymalismy sie pewnej konfencji nazewniczej:

$$[\boldsymbol{anchor\_type}][\boldsymbol{anchor\_num}][\boldsymbol{anchor\_double}] \quad | \quad [\boldsymbol{step\_function}] \quad + \quad [\boldsymbol{noise}]$$

dla przykladu Circle11(1/3)Dual(1/4)|Factor3+Noise ktore mozna przeczytac jako:

[3] anchorem nazywamy metode ktora pobiera sasiadow do przekazania informacji

$$
\begin{aligned}
[\text{anchor\_type}] &= Circle,\\
[\text{anchor\_num}] &= 11,\\
[\text{anchor\_number\_ratio}] &= 1/3,\\
[\text{anchor\_double}] &= True,\\
[\text{anchor\_distance\_ratio}] &= 1/4,\\
[\text{step\_function}] &= Factor3,\\
[\text{noise}] &= True
\end{aligned}
$$

4.1 Noise

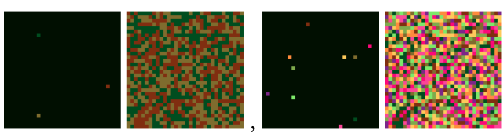

Fig. 1. Noise for 32x32

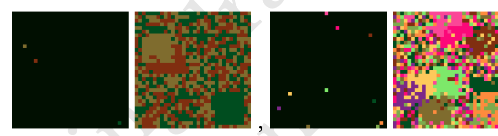

Fig. 2. Local Noise for 32x32

FIXME: figure z przykladami szumu (+local) i jak to wyglada i jak wygladalo instancja!

Zamiast zaczynac od pustej macierzy z seed-ami poczatkowymi mozna ja losowa uzupelnic szumem - tworzac przypadkowe short-cuty. Mozna tego dokonac osobnym kernelem ktory zostatnie wywolany przed wykonaniem glownej czesci algorytmu. Interpretacja jest taka ze pewne rejony ktora w JFA sa wypelnione zerami podczas pierwszych iteracji nie podejmuja zadnych decyzji. Uzupelniajac szumem moga one przypadkowo ustawic sie na prawidlowa wartosc i propagowac w kolejnej rundzie najlepsza wartosc w swoim otoczeniu (zgodnie ze stepem).

dowod kamila tutaj????????????????

4.1.1 Local Noise. przesunieta ciezkosci??? czyli srednia z wagami poprawila? **a moze uzaleznisc wage od rozmiaru pierwszego stepu?**

Mozna tez szum uzupelniac nie losowo tylko w otoczeniu. Wiec gdy w punkcie (x, y) wylosujemy losowego seed-a o wartosci (x_{rand}, y_{rand}) to wyliczamy nowa pozycje (x', y') ktora znajduje sie w polowie drogi w nastepujacy sposob: $x' = \frac{x+3x_{rand}}{4}$, analogicznie dla y'. Dodatkowo jesli (x', y') jest pusta to tez uzupelniamy to pole ta informacja. Nie przejmujemy sie wyscigiem w dostepie do danych. Nadpisania beda losowe - a szum tez.

4.2 Anchor Type

losowane punkty na okregu???

Zamiast pobierac informacje od 8 sasiadow o step size from grid points at $(x + i, y + j)$ where $i, j \in \{-step, 0, step\}$. Mozna zastosowac okrag - otwiera nam to nowe mozliwosci na swobodna modyfikacje ilosci punktow od ktorych bedziemy pobierac informacje. Naturalnie wydaje sie ze mala ilosc punktow w anchorze spowoduje wzrost bledu, a duza ilosc punktow spowoduje zmalenie bledu.

4.2.1 Anchor Number. Dlatego kolejnym parametrem bedzie mozliwosc kontrolowania ilosci punktow. Niestety nie rozwazalismy wariantu kwadratow o dowolnej ilosci punktow (poniewaz byly by to wielokrotnosci 2x2=4, 3x3=9, 4x4=16, 5x5=25) bo i tak nie dalo by sie wybrac uniformly tej wartosci. Dla okregu punkty sasiada (x_i, y_i) byly liczone nastepujaco:

$$x_i = x + \text{step} \cdot \cos\left(\frac{2\pi}{[\text{anchor\_num}]} \cdot i\right), y_i = y + \text{step} \cdot \sin\left(\frac{2\pi}{[\text{anchor\_num}]} \cdot i\right)$$

4.3 Anchor Double

Oprocz pojedynczego anchora, mozliwe jest uzycie podwojnej warstwy anchorow (czyli np. male kolko i wieksze). Idea za tym stojaca to ze male kolko wewnetrzne jest dokladne (dziala jak w JFA) - a wielkie zewnetrzne jest skautujace lub aby poprawic error wynikajacy np. z mniejszej ilosci anchor_num (w sumie to podobny mechanizm jak w Lookahead - wolny/szybki)

4.3.1 Anchor Distance Ratio. Parametr mowiacy o stosunku dlugosci step size od wewnetrzengo anchora do zewnetrznego.

4.3.2 Anchor Number Ratio. Parametr mowiacy o statusnku ilosci detektorow od wewnetrznego anchora do zewnetrznego.

4.4 Step Function

Gdy nasza informacja propaguje sie szybciej lub jest bardziej zageszczona dlatego sasiedzi szybciej dostaja prawidlowa informacje - to oznacza ze mozna skrocic ilosc round wykonania algorytmu.

Step size jak i ich ilosc mozna okreslic za pomoca 2 podstawowych parametrow: shape and number of points - z ktorych pozniej mozemy okreslic np. srednia gestosc. Zaimplementowalismy 2 warianty ktore sa uzaleznione jedynie od shape: defaultowy z JFA, z JFA o podstawie 3; oraz jeden uzaleniony od shape oraz od num: logstar. Jednak aby wygeneralizowac problem stworzylismy tez mozliwosc wygenerowania dowolnego polynomialu.

4.4.1 Special Polynomial. problem z Special - on overfituje przyklady zmieniajac 5 miejsce po przecinsku aby 2 zamienialo sie np. w 1

Implementacja nie jest wazna - chodzi o idea zwiazania shape oraz num. Oraz modyfikowanie wartosci, szybkosci spadku, ksztaltu (np. piloksztnego) - jakimis parametrami. Wada tego rozwiazania jest ze trzeba optymalizowac ta funkcje na calej dziedzinie (malej, duzej, gestej, zadkiej) - bo inaczej z overfituje ona ilosc krokow i wiekosc stepu pod rozmiar.

```
def mod_step_function__special(shape, num=None, config=None):
    # [EXAMPLE]
    # Special(1.51/0.92/0.92/1.08/0.42)
    # ------- A -- B -- C -- D -- X ---

    A = config["A"] # <1, 2>
    B = config["B"] # <0, 1>
    C = config["C"] # <0, 1>
    D = config["D"] # <1, 2>
    X = config["X"] # <0.2, 1>

    q = num / (shape[0] * shape[1])
    qm = ((shape[0] + shape[1]) / 2) * q**(1 / 2)
    S = B * qm + (1 - B) * (max(shape) / 2)
    St = math.log2(S)

    steps = []
    for i in range(1, int(X * St * 2), 1):
        f = round(1 / (D**(i**A) + i % max(1, int(C * St))), 4)
        ffm = int(f * S)
        if ffm >= 1:
```

```
        steps.append(ffm)
    if len(steps) == 0:
        return [1]

    return steps
```

5 RESULTS

przeniesc legende? JAKO OSOBNY PDF? i podac w tej sekcji - tak sie nie robi ale bylo by ok i czytelnie + wiecej miejsca na wykresy a przypadkow bedzie wiecej czy wykres loss oraz score dla przypadkow powinnien byc nalozony? albo polaczony subfigurem tak aby osie byly sync. i dalo sie porownac

performance plot[4]

UWAGE: usunac z tabelek PODOBNE ALGORYTMY i ich slabsze rezultaty

Shape / Density	Small (32-128)	Medium (256-448)	Large (512-1536)
Low (ρ=0.00005-0.001)	JFA	Blabla	?
High (ρ=0.01-0.1)	JFAStar	?	?

Table 3. Found in this paper state-of-the-art for specific domains.

5.1 Multi-domain Algorithm (JFAStar)

- **shapes**: {32x32, 64x64, 96x96, 128x128, 256x256, 320x320, 384x384, 448x448, 512x512, 768x768, 1024x1024, 1536x1536}
- **cases**:
  - gen_uniform: seeds=1,
  - gen_uniform: seeds=3,
  - gen_uniform: density=0.0001,
  - gen_uniform: density=0.001,
  - gen_uniform: density=0.01,
  - gen_uniform: density=0.02,
  - gen_uniform: density=0.03,
  - gen_uniform: density=0.04,
  - gen_uniform: density=0.05,
  - gen_uniform: density=0.1,

[4]wykres zostal zrobiony poprzez posortowanie scorow - dzieki temu widac roznice w przyroscie i latwo dostrzec ktory algorytm ma najwyzszy score lub jaka ma chaktersytyke (np. jest bardzo skuteczny dla waskiej grupy przykladow)

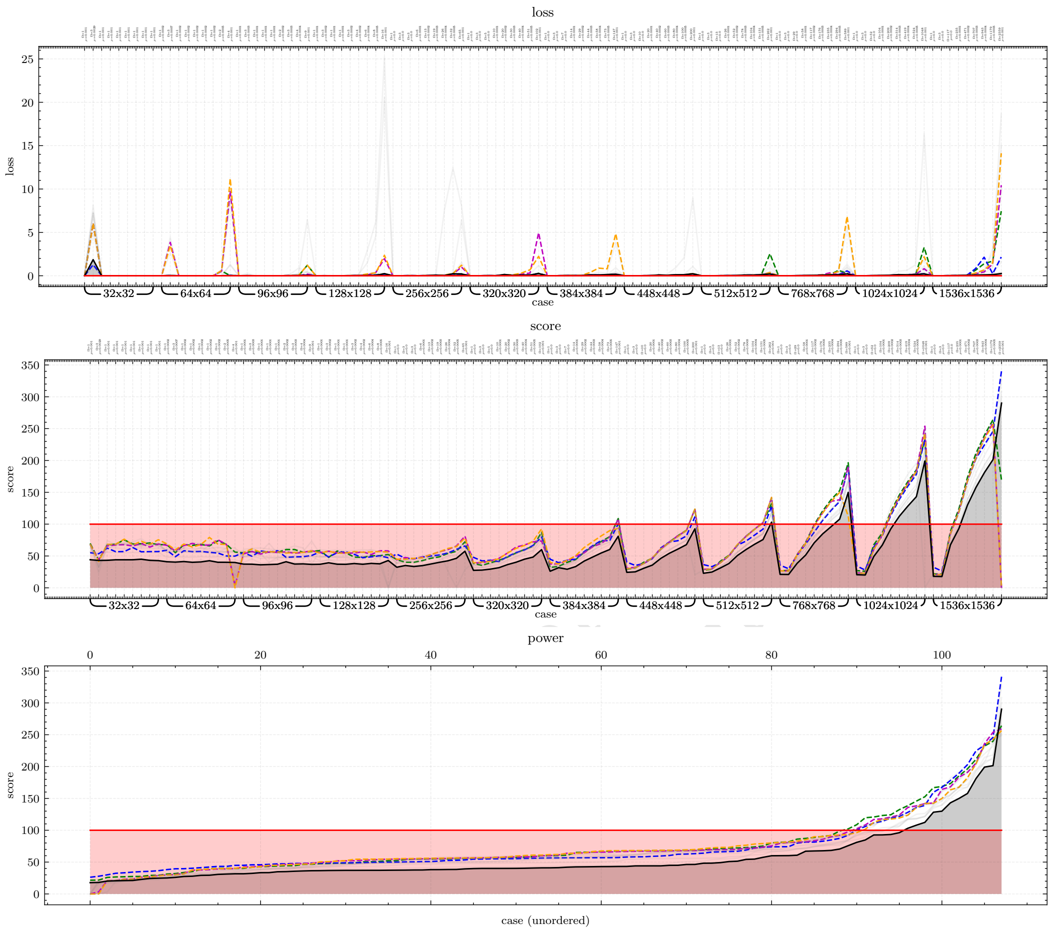

Fig. 3. bla bla bla

Algorithm	ρ=0.0003	ρ=0.0002	ρ=0.0003	ρ=0.0004	ρ=0.0008	Score
Square|Special(1.92/0.9/0.68/1.2/0.67)+Noise	40.4	51.1	74.8	95.1	**113.6**	74
Square|Star+Noise	44.5	50.5	71.2	87.9	**115.8**	73
Square|Special(1.98/0.8/0.95/1.2/0.46)+Noise	40.8	51.5	74.7	95.5	**102.6**	72
Square|Special(1.47/0.68/0.75/1.52/0.41)+Noise	41	51.9	75.5	96.7	98.3	69
Square|Special(1.46/0.81/0.68/1.33/0.5)+Noise	39.6	50.9	73	88.1	**112.1**	69
Square|Special(1.98/0.69/0.74/1.22/0.49)+Noise	37.4	49.5	71.2	91.7	**102.6**	66
Square|Special(1.14/0.8/0.61/1.38/0.62)+Noise	34.6	44.9	64.7	80.4	**108.7**	66
SquareDual(1/3)|Special(1.56/0.88/0.57/1.64/0.39) +Noise	39.1	48.5	67.5	85.7	98.2	65
Square|Special(1.97/0.79/0.61/1.24/0.61)+Noise	37.9	49.5	72.8	93.4	88.2	64
Square|Special(1.92/0.55/0.61/1.1/0.53)+Noise	33.7	41.8	60.9	77.5	**105.4**	63
Square|Special(1.83/0.69/0.81/1.24/0.45)+Noise	38.3	49.5	72.5	92.7	**102.5**	62
Square|Factor3+Noise	31	39.4	58.9	76.2	**105.3**	61
Square|Special(1.84/0.65/0.84/1.26/0.52)+Noise	37.2	48.8	70.3	89.7	**101.9**	60
Square|Special(1.8/0.65/0.58/1.27/0.59)+Noise	36.9	46	67.5	86.8	98.0	59
Square|Special(1.94/0.64/0.8/1.28/0.4)+Noise	40.1	52	77.7	96.3	88.2	58
Square|Special(1.82/0.71/0.73/1.2/0.42)+Noise	40.4	51.5	75.1	94.4	82.5	58
Square|Special(1.33/0.5/0.7/1.26/0.46)+Noise	35	46.1	65.8	80.9	81.3	57
SquareDual(1/3)|Special(1.71/0.06/0.95/1.46/0.34) +Noise	32	38	54.2	68.8	92.2	56
Square|Special(1.93/0.64/0.92/1.21/0.98)+Noise	34.6	43.9	65.6	85.4	96.7	56
JFA (original)	30.2	37.2	54.2	69.2	94.4	56
SquareDual(1/3)|Special(1.51/0.46/0.97/1.88/0.57) +Noise	31.7	39.1	56.7	72.8	83.4	55
SquareDual(3/4)|Star+Noise	34.8	38.6	52.9	65.7	84.8	54
SquareDual(1/3)|Star+Noise	35.4	38	52.5	65.1	83.5	54
SquareDual|Star+Noise	34.6	38	52	65.2	83.3	54
SquareDual(1/3)|Special(1.2/0.9/0.84/1.96/0.44) +Noise	38.6	47.2	65.7	73.5	95.8	53
Square|Default+Noise	26.6	33.6	49.9	64.1	88.7	52
Circle10(1/4)|Special(1.58/0.9/0.64/1.16/0.6)+Noise	29.3	34.7	46.8	57.1	72.9	47
SquareDual(1/3)|Special(1.45/0.85/0.74/1.95/0.34) +LNoise	40.2	44.2	45.2	54.6	56.7	45
Circle11(1/3)|Star+Noise	30	32.5	44.5	51.7	67.4	44
SquareDual|Special(1.48/0.39/0.84/1.34/0.72) +Noise	24	29.9	42.8	54.7	74.6	44
Circle11(3/4)Dual(2/3)|Factor3+Noise	24.7	30	43.1	56.1	69.9	44
Circle9(1/4)|Star+Noise	31.8	35.1	47.6	49.9	61.9	44
Circle9(3/4)|Star+Noise	31.9	34.6	47.1	50.8	61.8	44
Circle12(1/3)|Star+Noise	28.6	32	42.7	51.2	65.5	43
Circle14(1/4)|Special(1.38/0.71/0.93/1.77/0.95) +Noise	24.3	29.4	41.5	51.5	62.2	41
Square|Special(1.5[illegible]/0.7/0.65/1.29/0.43)+Noise	40.9	51.6	73.7	92.9	73.2	40
SquareDual(3/4)|Default	22.5	27.6	39.1	49.7	66.9	40
SquareDual|Default	22.6	27.3	38.8	49.2	65.7	40

5.2 Specific-domain Algorithm

5.2.1 *Small Shape: 32x32, 64x64, 96x96, 128x128.*

5.2.2 *Medium Shape: 256x256, 320x320, 384x384, 448x448.*

5.2.3 *Large Shape: 512x512, 768x768, 1024x1024, 1536x1536.*

5.2.4 *Low Density.*

5.2.5 *High Density.*

5.3 Ensemble Across Domains (Vorotron)

bla bla

5.4 Objectives

naprawic generowanie tego wykresu napisac co nie moze byc uzyte z czym? czyli co ma wplyw na co (w sumie to najwazniejsze mialo byc w pracy)

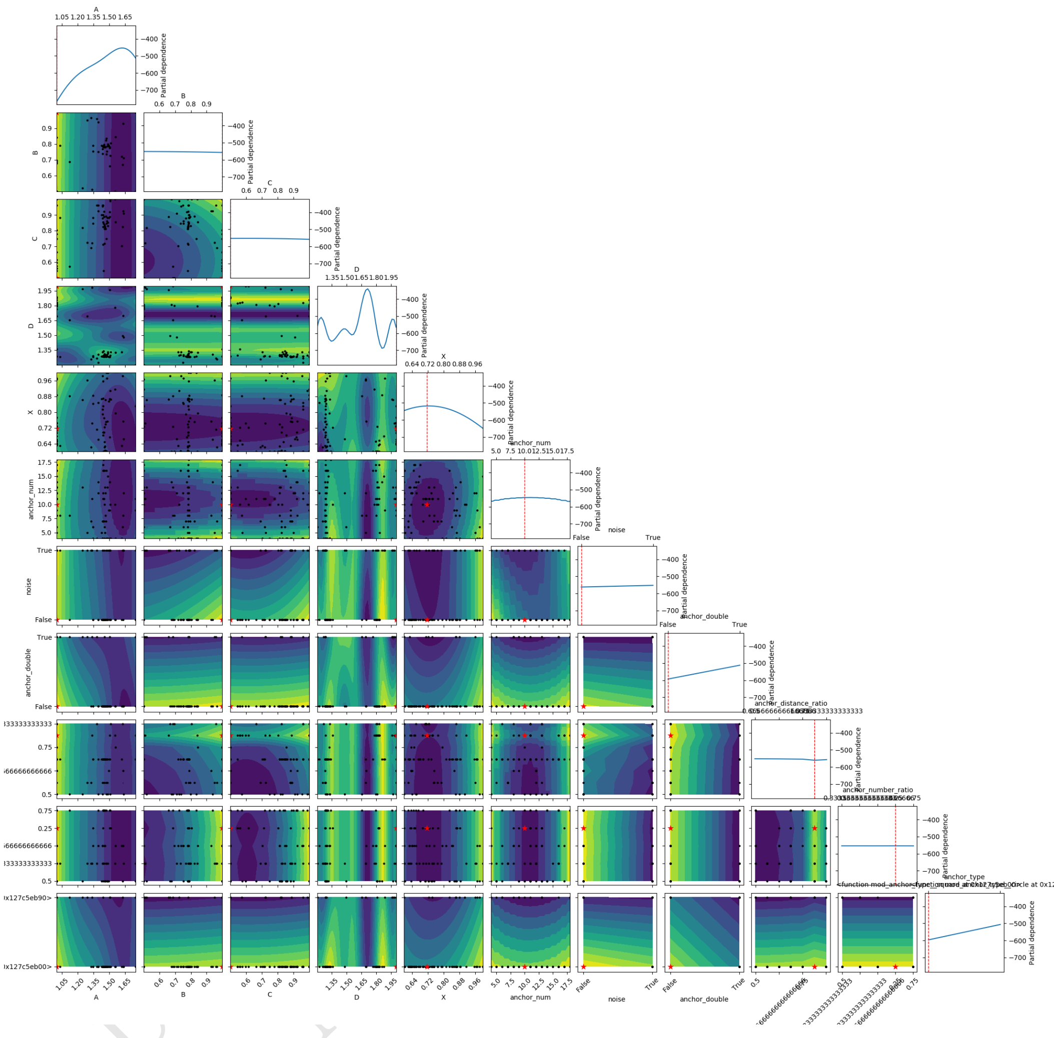

Fig. 4. bla bla bla

6 PRACTICAL USAGE

polaczyc z Conclusions

Jest wiele projektow ktore potrzebuje DT lub voronoi-a. Jedyne dwa praktyczne przyklady z tej pracy to SOTA dla JFA - czyli JFAstar, oraz praktyczny Ensemble (uwzgledniajacy np. bruteforce dla malych instancji).

7 CONCLUSIONS

This paper presents the GPU's effective, almost constant, algorithm for calculating the Euclidean distance transform (DT) approximation for 2D and higher dimensional images. As mentioned in [1], it remains challenging to balance the workload in such an approach. *Vorotron* does not explicitly solve this issue but, by constructing an alternative solution utilizing random shortcuts and parameter estimation, it makes it a reasonable approximation. In practice, such a constant time algorithm is useful in many interactive applications, such as tessellations, rendering, and image processing, involving [7].

8 ACKNOWLEDGEMENTS

Dziekuje swojemu psu!Dziekuje swojemu psu! Dziekuje swojemu psu!

REFERENCES

[1] Thanh-Tung Cao, Ke Tang, Anis Mohamed, and Tiow-Seng Tan. 2010. Parallel banding algorithm to compute exact distance transform with the GPU. In *Proceedings of the 2010 ACM SIGGRAPH symposium on Interactive 3D Graphics and Games*. 83–90.

[2] Francisco de Assis Zampirolli and Leonardo Filipe. 2017. A fast CUDA-based implementation for the Euclidean distance transform. In *2017 International Conference on High Performance Computing & Simulation (HPCS)*. IEEE, 815–818.

[3] Ian Fischer and Craig Gotsman. 2006. Fast approximation of high-order Voronoi diagrams and distance transforms on the GPU. *Journal of Graphics Tools* 11, 4 (2006), 39–60.

[4] Kenneth E Hoff III, John Keyser, Ming Lin, Dinesh Manocha, and Tim Culver. 1999. Fast computation of generalized Voronoi diagrams using graphics hardware. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*. 277–286.

[5] Takumi Honda, Shinnosuke Yamamoto, Hiroaki Honda, Koji Nakano, and Yasuaki Ito. 2017. Simple and fast parallel algorithms for the Voronoi map and the Euclidean distance map, with GPU implementations. In *2017 46th International Conference on Parallel Processing (ICPP)*. IEEE, 362–371.

[6] Manduhu Manduhu and Mark W Jones. 2019. A work efficient parallel Algorithm for exact Euclidean distance transform. *IEEE Transactions on Image Processing* 28, 11 (2019), 5322–5335.

[7] Guodong Rong and Tiow-Seng Tan. 2006. Jump flooding in GPU with applications to Voronoi diagram and distance transform. In *Proceedings of the 2006 symposium on Interactive 3D graphics and games*. 109–116.

[8] Guodong Rong and Tiow-Seng Tan. 2007. Variants of jump flooding algorithm for computing discrete Voronoi diagrams. In *4th International Symposium on Voronoi Diagrams in Science and Engineering (ISVD 2007)*. IEEE, 176–181.

[9] Jens Schneider, Martin Kraus, and Rüdiger Westermann. 2009. GPU-based real-time discrete Euclidean distance transforms with precise error bounds.. In *VISAPP (1)*. 435–442.

[10] Avneesh Sud, Naga Govindaraju, Russell Gayle, and Dinesh Manocha. 2006. Interactive 3D distance field computation using linear factorization. In *Proceedings of the 2006 symposium on Interactive 3D graphics and games*. 117–124.