Inter-University Research Institute Corporation
Research Organization of Information and Systems
National Institute of Informatics

Revitalizing Research on Approximate String Matching Algorithms

Philip Wellnitz

National Institute of Informatics

Based on joint works with Karl Bringmann, Alejandro Cassis, Panagiotis Charalampopoulos, Tomasz Kociumaka, Marvin Künnemann, and Jakob Nogler.

Research Focus: Theoretical Guarantees for Fundamental Problems

Approximate String Matching

[BK**W**,SODA’19]
[CK**W**,FOCS’20,’22,’25]
[CK**W**,FOCS’23]
[NK**W**,STOC’24,
SODA’25]

Counting Structures in Networks

[DR**W**,ICALP’19][R**W**,SODA’20]
[RS**W**,FOCS’20,ICALP’21]
[DRS**W**,MFCS’19]
[DM**W**,STOC’24,
SODA’25]

Dense Subset Sum

[B**W**,SODA’20]

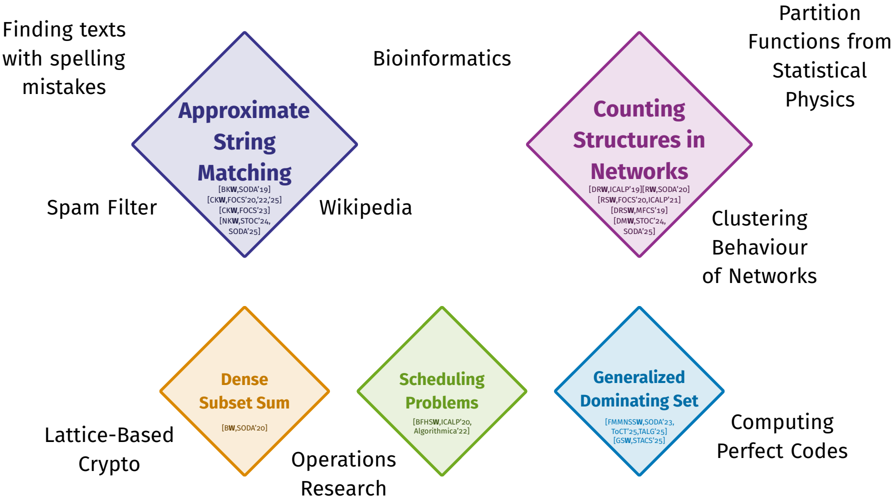
Scheduling Problems

[BFHS**W**,ICALP’20,
Algorithmica’22]

Generalized Dominating Set

[FMMNSS**W**,SODA’23,
ToCT’25,TALG’25]
[GS**W**,STACS’25]

Research Focus: Theoretical Guarantees for Fundamental Problems of Practical Relevance



Approximate String Matching
[BK**W**,SODA'19]
[CK**W**,FOCS'20,'22,'25]
[CK**W**,FOCS'23]
[NK**W**,STOC'24,
SODA'25]

Finding texts with spelling mistakes

Bioinformatics

Spam Filter

Wikipedia

Counting Structures in Networks
[DR**W**,ICALP'19][R**W**,SODA'20]
[RS**W**,FOCS'20,ICALP'21]
[DRS**W**,MFCS'19]
[DM**W**,STOC'24,
SODA'25]

Partition Functions from Statistical Physics

Clustering Behaviour of Networks

Dense Subset Sum
[B**W**,SODA'20]

Lattice-Based Crypto

Scheduling Problems
[BFHS**W**,ICALP'20,
Algorithmica'22]

Operations Research

Generalized Dominating Set
[FMMNSS**W**,SODA'23,
ToCT'25,TALG'25]
[GS**W**,STACS'25]

Computing Perfect Codes

Inter-University Research Institute Corporation
Research Organization of Information and Systems
National Institute of Informatics

Approximate String Matching

An Example

...

❀❀ ❀ ❀ ❀❀❀❀ ❀❀❀❀❀❀❀❀ ❀❀ ❀❀❀❀ ❀❀ ❀❀❀❀ ❀❀❀ ❀❀❀ ❀❀❀ ❀ ❀❀❀ ❀❀❀ ❀ ❀❀ (TU Kaiserslautern). ❀❀❀❀ ❀ ❀❀❀❀❀❀❀❀❀❀ ❀❀❀❀❀❀ ❀ ❀❀❀❀❀❀❀❀ ❀ ❀❀ ❀❀ ❀ ❀ ❀ (Simons Institute and UC Berkeley), ❀ ❀❀❀ ❀❀❀ (University of Salzburg), ❀❀❀❀❀❀ ❀ (University of Salzburg). ❀ ❀ ❀❀❀❀ ❀❀❀❀❀❀❀❀❀ ❀❀ ❀❀ ❀ ❀❀ ❀❀❀ (University of Pennsylvania), ❀❀ ❀ ❀ ❀ ❀ (University of Pennsylvania), ❀❀ ❀ (University of Pennsylvania). ❀❀❀ ❀❀ ❀❀❀ ❀❀ ❀ ❀❀❀❀ ❀ ❀ ❀❀ ❀❀❀❀❀ ❀❀❀❀❀❀❀❀❀❀❀❀❀❀ (Reichman University, Herzliya, Israel and Birkbeck, University of London), ❀❀❀❀ ❀❀❀❀❀❀❀ (UC Berkeley and Max Planck Institute for Informatics, SIC, Saarbrücken, Germany), ❀ ❀❀ ❀❀ (Max Planck Institute for Informatics, SIC, Saarbrücken, Germany). ❀❀ ❀❀ ❀❀❀❀❀ ❀ ❀❀❀❀❀❀❀❀ ❀❀❀❀❀❀❀❀❀❀❀ ❀ ❀❀ ❀ ❀❀ (National Institute of Informatics). ❀ ❀❀❀ ❀ ❀ ❀ ❀❀❀❀❀❀❀ ❀❀❀❀❀❀❀❀❀ ❀❀❀❀❀❀❀❀ ❀ ❀❀❀❀ (UniversitÃ© Paris CitÃ©, CNRS, IRIF, F-75013, Paris, France); ❀❀❀❀ ❀ ❀ ❀❀❀ (Universitat PolitÃ¨cnica de Catalunya)

...

...

❀ ❀ ❀❀❀❀❀ ❀ ❀❀❀ ❀ ❀ ❀❀❀❀❀❀❀❀ ❀ ❀❀❀❀❀ ❀❀ ❀ ❀❀❀❀❀❀❀❀ ❀ ❀❀❀ ❀❀ ❀ ❀❀❀ ❀❀ ❀❀❀❀ ❀ ❀ ❀ (ETH Zurich); ❀❀ ❀❀ ❀❀❀❀❀ ❀ ❀❀ (Toyota Technological Institute at Chicago) ❀❀❀ ❀❀❀❀❀❀ ❀❀❀ ❀❀❀❀❀❀ ❀❀❀❀❀❀ ❀ ❀❀❀ ❀❀ ❀❀❀ ❀ ❀❀❀❀ (Merton College, University of Oxford, United Kingdom); ❀❀ ❀ ❀❀❀ ❀ (Mathematical Institute, University of Bonn, Germany); ❀ ❀❀ ❀❀ (Max Planck Institute for Informatics, Saarland Informatics Campus (SIC), Saarbr|cken, Germany) ❀❀❀ ❀❀❀❀ ❀ ❀❀ ❀ ❀❀❀❀❀❀❀ ❀ ❀❀❀❀ ❀❀❀ ❀ ❀❀❀❀❀❀❀ ❀❀ ❀❀❀❀❀ ❀ (The Academic College of Tel Aviv-Yafo), ❀❀❀❀ ❀❀❀❀❀❀❀ (UC Berkeley), ❀❀ ❀❀ ❀❀❀ (Weizmann Institute of Science), ❀❀❀❀❀ ❀ (University of California San Diego). ❀ ❀❀❀❀❀ ❀❀❀❀ ❀❀❀❀❀❀❀❀ ❀❀ ❀❀❀❀❀❀ ❀ ❀❀ ❀ ❀❀ ❀ ❀❀ ❀ ❀❀❀ ❀ ❀❀❀❀❀❀❀❀ ❀❀ ❀ ❀❀❀❀❀❀❀ ❀❀❀ ❀❀ ❀ (CISPA Helmholtz Center for Information Security, Saarbrücken); ❀❀ ❀ ❀ (Simon Fraser University); ❀❀❀ ❀❀❀ ❀ (Duke University); ❀ ❀❀❀❀❀❀ ❀❀ (CISPA Helmholtz Center for Information Security, Saarbrücken); ❀ ❀❀ ❀❀ (Max Planck Institute for Informatics, SIC, Saarbrücken)

...

...

❀❀❀❀ ❀ ❀❀❀❀❀ ❀ ❀❀ ❀❀❀❀ ❀❀❀❀❀❀❀❀❀ ❀❀ ❀ ❀ (Alibaba Group); ❀❀ ❀❀ ❀ (Microsoft Research); ❀ ❀❀❀ ❀ (Washington University in St. Louis); ❀ ❀ (Columbia University); ❀❀ ❀❀❀ (University of Chicago) ❀ ❀ ❀❀ ❀❀❀ ❀❀ ❀❀❀❀ ❀❀❀❀❀❀❀❀❀ ❀❀❀❀❀❀❀❀❀❀❀❀ ❀❀❀❀❀❀ ❀ ❀❀❀ ❀❀❀ (Stony Brook University); ❀❀❀❀ ❀❀❀❀❀❀ (Max Planck Institute for Informatics, Saarbruecken) ❀❀❀❀ ❀❀ ❀❀❀❀❀❀❀❀❀ ❀❀ ❀ ❀❀❀ ❀❀❀ ❀❀❀ ❀ ❀❀❀❀ ❀ ❀❀❀❀❀❀ ❀ ❀❀❀❀ ❀ ❀❀ ❀❀ ❀❀❀❀ ❀❀❀ ❀❀❀ ❀❀ ❀❀❀❀❀❀❀ (Univ. Bordeaux, CNRS, Bordeaux INP, LaBRI, France); ❀❀ ❀❀❀❀ ❀ ❀❀ (University of Warsaw, Poland); ❀❀❀ ❀ ❀❀ (Saarland University and Max Planck Institute for Informatics, SaarbrÃ¼cken, Germany); ❀ ❀ ❀❀❀❀❀ ❀ ❀ (University of Warsaw, Poland) ❀❀❀❀ ❀❀❀❀ ❀❀❀ ❀❀ ❀❀ ❀❀ ❀❀ ❀❀❀ ❀❀ ❀ ❀ ❀❀❀❀ ❀ ❀ ❀❀ ❀❀❀❀❀ ❀ ❀ (ETH ZÃ¼rich); ❀❀ ❀❀ ❀❀❀❀ ❀ ❀❀❀ ❀ (TU Berlin) ❀ ❀ ❀ ❀ ❀ ❀❀❀❀❀ ❀ ❀❀❀❀❀❀ ❀❀❀❀❀❀❀❀ ❀❀❀❀ ❀❀❀❀❀❀ (Carnegie Mellon University)

...

An Example

...

✿✿ ✿ ✿ ✿✿✿✿ ✿✿✿✿✿✿✿ ✿✿ ✿✿✿✿ ✿✿✿✿ ✿✿✿✿ ✿✿✿ ✿✿✿ ✿✿✿ ✿ ✿✿✿

✿✿✿ ✿ ✿✿ (TU Kaiserslautern).

✿✿✿✿ ✿ ✿✿✿✿✿✿✿✿✿✿ ✿✿✿✿✿✿ ✿ ✿✿✿✿✿✿✿✿ ✿

✿✿ ✿✿ ✿ ✿ ✿ (Simons Institute and UC Berkeley), ✿ ✿✿✿ ✿✿✿ (University of Salzburg), ✿✿✿✿✿✿ ✿ (University of Salzburg).

✿ ✿ ✿✿✿✿ ✿✿✿✿✿✿✿✿ ✿✿ ✿✿ ✿ ✿✿✿✿✿

(University of Pennsylvania), ✿✿ ✿ ✿ ✿ ✿ (University of Pennsylvania), ✿✿ ✿ (University of Pennsylvania).

✿✿✿ ✿✿ ✿✿✿ ✿✿ ✿ ✿✿✿✿ ✿ ✿

✿✿ ✿✿✿✿✿ ✿✿✿✿✿✿✿✿✿✿✿✿✿✿ (Reichman University, Herzliya, Israel and Birkbeck, University of London), ✿✿✿✿ ✿✿✿✿✿✿ (UC Berkeley and Max Planck Institute for Informatics, SIC, **Saarbrücken**, Germany), ✿ ✿✿ ✿✿ (Max Planck Institute for Informatics, SIC, **Saarbrücken**, Germany).

✿✿ ✿✿ ✿✿✿✿✿ ✿ ✿✿✿✿✿✿✿✿ ✿✿✿✿✿✿✿✿✿✿

✿ ✿✿ ✿ ✿✿ (National Institute of Informatics).

✿ ✿✿✿ ✿ ✿ ✿ ✿✿✿✿✿✿ ✿✿✿✿✿✿✿✿✿

✿✿✿✿✿✿✿ ✿ ✿✿✿✿ (UniversitÃ© Paris CitÃ©, CNRS, IRIF, F-75013, Paris, France); ✿✿✿✿ ✿ ✿ ✿✿✿ (Universitat PolitÃ¨cnica de Catalunya)

...

...

✿ ✿ ✿✿✿✿✿ ✿ ✿✿✿ ✿ ✿ ✿✿✿✿✿✿✿✿ ✿ ✿✿✿✿✿ ✿ ✿✿✿✿✿✿✿ ✿ ✿✿✿ ✿✿ ✿ ✿✿✿ ✿✿

✿✿✿✿ ✿ ✿ ✿ (ETH Zurich); ✿✿ ✿✿ ✿✿✿✿✿✿ ✿ ✿✿ (Toyota Technological Institute at Chicago)

✿✿✿ ✿✿✿✿✿✿ ✿✿✿ ✿✿✿✿✿✿ ✿✿✿✿✿✿ ✿ ✿✿✿ ✿✿ ✿✿✿ ✿

✿✿✿✿ (Merton College, University of Oxford, United Kingdom); ✿✿ ✿ ✿✿✿ ✿ (Mathematical Institute, University of Bonn, Germany); ✿ ✿✿ ✿✿ (Max Planck Institute for Informatics, Saarland Informatics Campus (SIC), Saarbr|cken, Germany)

✿✿✿ ✿✿✿✿ ✿ ✿✿ ✿ ✿✿✿✿✿✿ ✿ ✿✿✿✿ ✿✿✿ ✿ ✿✿✿✿✿✿✿

✿✿ ✿✿✿✿✿ ✿ (The Academic College of Tel Aviv-Yafo), ✿✿✿✿ ✿✿✿✿✿✿✿ (UC Berkeley), ✿✿✿ ✿✿✿ (Weizmann Institute of Science), ✿✿✿✿ ✿ (University of California San Diego).

✿ ✿✿✿✿✿ ✿✿✿✿ ✿✿✿✿✿✿✿ ✿✿ ✿✿✿✿✿✿ ✿ ✿✿ ✿ ✿✿ ✿ ✿✿ ✿ ✿✿✿ ✿

✿✿✿✿✿✿✿ ✿✿ ✿ ✿✿✿✿✿✿ ✿✿✿ ✿✿ ✿ (CISPA Helmholtz Center for Information Security, **Saarbrücken**); ✿✿ ✿ ✿ (Simon Fraser University); ✿✿✿ ✿✿✿ ✿ (Duke University); ✿ ✿✿✿✿✿ ✿✿ (CISPA Helmholtz Center for Information Security, **Saarbrücken**); ✿ ✿✿ ✿✿ (Max Planck Institute for Informatics, SIC, **Saarbrücken**)

...

...

✿✿✿✿ ✿ ✿✿✿✿✿ ✿ ✿✿ ✿✿✿✿ ✿✿✿✿✿✿✿✿✿

✿✿ ✿ ✿ (Alibaba Group); ✿✿ ✿✿ ✿ (Microsoft Research); ✿ ✿✿✿ ✿ (Washington University in St. Louis); ✿ ✿ (Columbia University); ✿✿ ✿✿✿ (University of Chicago)

✿ ✿ ✿✿ ✿✿✿ ✿✿ ✿✿✿✿ ✿✿✿✿✿✿✿✿ ✿✿✿✿✿✿✿✿✿✿✿ ✿✿✿✿✿✿ ✿

✿✿✿ ✿✿✿ (Stony Brook University); ✿✿✿✿ ✿✿✿✿✿✿ (Max Planck Institute for Informatics, Saarbruecken)

✿✿✿✿ ✿✿ ✿✿✿✿✿✿✿✿ ✿✿ ✿ ✿✿✿ ✿✿✿ ✿✿✿ ✿ ✿✿✿✿ ✿ ✿✿✿✿✿✿ ✿ ✿✿✿✿ ✿ ✿✿ ✿✿ ✿✿✿✿ ✿✿✿ ✿✿✿

✿✿ ✿✿✿✿✿✿ (Univ. Bordeaux, CNRS, Bordeaux INP, LaBRI, France); ✿✿ ✿✿✿✿ ✿ ✿✿ (University of Warsaw, Poland); ✿✿✿ ✿ ✿✿ (Saarland University and Max Planck Institute for Informatics, SaarbrÃ¼cken, Germany); ✿ ✿ ✿✿✿✿✿ ✿ ✿ (University of Warsaw, Poland)

✿✿✿✿ ✿✿✿✿ ✿✿✿ ✿✿ ✿✿ ✿✿ ✿✿ ✿✿✿✿ ✿ ✿ ✿✿✿✿ ✿ ✿ ✿✿

✿✿✿✿✿ ✿ ✿ (ETH ZÃ¼rich); ✿✿ ✿✿ ✿✿✿✿ ✿ ✿✿✿ ✿ (TU Berlin)

✿ ✿ ✿ ✿ ✿ ✿✿✿✿✿ ✿

✿✿✿✿✿✿ ✿✿✿✿✿✿✿ ✿✿✿✿ ✿✿✿✿✿✿ (Carnegie Mellon University)

...

An Example

…

✿✿ ✿ ✿ ✿✿✿✿ ✿✿✿✿✿✿✿✿ ✿✿ ✿✿✿✿ ✿-✿ ✿✿✿✿ ✿✿✿ ✿✿✿ ✿✿✿ ✿ ✿✿✿ ✿✿✿ ✿ ✿✿ (TU Kaiserslautern). ✿✿✿✿ ✿ ✿✿✿✿✿✿✿✿✿ ✿✿✿✿✿✿ ✿ ✿✿-✿✿✿✿✿✿ ✿ ✿✿ ✿✿ ✿ ✿ ✿ (Simons Institute and UC Berkeley), ✿ ✿✿✿ ✿✿✿ (University of Salzburg), ✿✿✿✿✿-✿ ✿ (University of Salzburg). ✿ ✿ ✿✿✿✿ ✿✿✿✿✿✿✿✿ ✿✿ ✿✿ ✿ ✿✿ ✿✿✿ (University of Pennsylvania), ✿✿ ✿ ✿ ✿ ✿ (University of Pennsylvania), ✿✿ ✿ (University of Pennsylvania). ✿✿✿ ✿✿ ✿✿✿ ✿✿ ✿ ✿✿✿✿ ✿ ✿ ✿✿ ✿✿✿✿✿ ✿✿✿✿✿✿✿✿✿✿✿✿✿✿ (Reichman University, Herzliya, Israel and Birkbeck, University of London), ✿✿✿✿ ✿✿✿✿✿✿ (UC Berkeley and Max Planck Institute for Informatics, SIC, **Saarbrücken**, Germany), ✿ ✿✿ ✿✿ (Max Planck Institute for Informatics, SIC, **Saarbrücken**, Germany). ✿✿ ✿✿ ✿✿✿✿✿ ✿ ✿✿✿✿✿✿✿✿ ✿✿✿-✿✿✿✿✿✿✿ ✿ ✿✿ ✿ ✿✿ (National Institute of Informatics). ✿ ✿✿✿ ✿ ✿ ✿ ✿✿✿✿✿✿ ✿✿✿✿✿✿✿✿ ✿✿✿✿✿✿✿ ✿ ✿✿✿✿ (UniversitÃ© Paris CitÃ©, CNRS, IRIF, F-75013, Paris, France); ✿✿✿✿ ✿ ✿ ✿-✿✿ (Universitat PolitÃ¨cnica de Catalunya)

…

…

✿ ✿ ✿✿✿✿✿ ✿ ✿✿✿ ✿ ✿ ✿✿✿✿✿✿✿✿ ✿ ✿-✿✿ ✿ ✿✿✿✿✿✿✿ ✿ ✿✿✿ ✿✿ ✿ ✿✿✿ ✿✿ ✿✿✿✿ ✿ ✿ ✿ (ETH Zurich); ✿✿ ✿✿ ✿✿✿✿-✿ ✿✿ (Toyota Technological Institute at Chicago) ✿✿✿ ✿✿✿✿✿✿ ✿✿✿ ✿✿✿✿✿✿ ✿✿✿✿✿-✿ ✿✿✿ ✿✿ ✿✿✿ ✿ ✿✿✿✿ (Merton College, University of Oxford, United Kingdom); ✿✿ ✿ ✿✿✿ ✿ (Mathematical Institute, University of Bonn, Germany); ✿ ✿✿ ✿-✿ (Max Planck Institute for Informatics, Saarland Informatics Campus (SIC), **Saarbr|cken**, Germany) ✿✿✿ ✿✿✿✿ ✿ ✿✿ ✿ ✿✿✿✿✿✿ ✿ ✿✿✿-✿✿✿ ✿ ✿✿✿✿✿✿✿ ✿✿ ✿✿✿✿✿ ✿ (The Academic College of Tel Aviv-Yafo), ✿✿✿✿ ✿✿✿✿✿✿✿ (UC Berkeley), ✿ -✿✿ ✿✿✿ (Weizmann Institute of Science), ✿-✿✿✿ ✿ (University of California San Diego). ✿ ✿✿✿✿✿ ✿✿✿✿ ✿✿✿✿✿✿✿ ✿✿ ✿✿-✿✿✿✿ ✿ ✿✿ ✿ ✿✿ ✿ ✿✿ ✿ ✿✿✿ ✿ ✿✿✿✿✿✿✿ ✿✿ ✿ ✿✿✿✿✿✿ ✿✿✿ ✿✿ -✿ (CISPA Helmholtz Center for Information Security, **Saarbrücken**); ✿✿ ✿ ✿ (Simon Fraser University); ✿✿✿ ✿✿✿ ✿ (Duke University); ✿ ✿✿✿✿✿ ✿-✿ (CISPA Helmholtz Center for Information Security, **Saarbrücken**); ✿ ✿✿ ✿✿ (Max Planck Institute for Informatics, SIC, **Saarbrücken**)

…

…

✿✿✿✿ ✿ ✿✿✿✿✿ ✿ ✿✿ ✿✿✿✿ ✿✿✿✿✿-✿✿✿✿ ✿✿ ✿ ✿ (Alibaba Group); ✿✿ ✿✿ ✿ (Microsoft Research); ✿ ✿✿✿ ✿ (Washington University in St. Louis); ✿ ✿ (Columbia University); ✿✿ ✿✿✿ (University of Chicago) ✿ ✿ ✿✿ ✿✿✿ ✿✿ ✿✿✿✿ ✿✿✿✿✿✿-✿✿ ✿✿✿✿✿✿✿✿✿✿✿✿ ✿✿✿✿✿✿ ✿ ✿✿✿ ✿✿✿ (Stony Brook University); ✿✿✿✿ ✿-✿✿✿✿✿ (Max Planck Institute for Informatics, **Saarbruecken**) ✿✿✿✿ ✿✿ ✿✿✿✿✿✿✿✿ ✿✿ ✿ ✿✿✿ ✿✿-✿ ✿✿✿ ✿ ✿✿✿✿ ✿ ✿✿✿✿✿ ✿ ✿✿✿✿ ✿ -✿✿ ✿✿ ✿✿✿✿ ✿✿✿ ✿✿✿ ✿✿ ✿✿✿✿✿✿✿ (Univ. Bordeaux, CNRS, Bordeaux INP, LaBRI, France); ✿✿ ✿✿✿✿✿ ✿ ✿-(University of Warsaw, Poland); ✿✿✿ ✿ ✿✿-(Saarland University and Max Planck Institute for Informatics, **SaarbrÃ¼cken**, Germany); ✿ ✿ ✿✿-✿✿✿ ✿ ✿ (University of Warsaw, Poland) ✿✿✿✿ ✿✿✿✿ ✿✿✿ ✿✿ ✿✿ ✿✿ ✿✿ ✿✿-✿✿ ✿ ✿ ✿✿✿✿ ✿ ✿ ✿✿ ✿✿✿✿✿ ✿ ✿ (ETH ZÃ¼rich); ✿✿ ✿✿ ✿✿-✿✿ ✿ ✿✿✿ ✿ (TU Berlin) ✿ ✿ ✿ ✿ ✿ ✿✿✿✿✿ ✿ ✿✿✿✿✿✿ ✿✿✿✿✿✿✿ ✿✿✿✿ ✿✿✿✿✿✿ (Carnegie Mellon University)

…

An Example

Task: Find **Saarbrücken** in a text.

An Example

Task: Find **Saarbrücken** in a text.

Or Saarbruecken.

An Example

Task: Find **Saarbrücken** in a text.

Or Saarbruecken. Or Sarrebruck.

An Example

Task: Find **Saarbrücken** in a text.

Or Saarbruecken. Or Sarrebruck. Or Saarbrucken, Saarbr|cken, SaarbrÃ¼cken, ...

The Approximate String Matching Problem

early 1980’s

Approximate String Matching

Given a text T, a pattern P, and an integer k, identify the (starting positions of) substrings of T that are at **edit distance** of at most k to P.

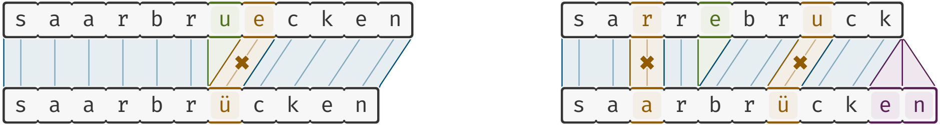
The Approximate String Matching Problem

early 1980’s

Approximate String Matching

Given a text T, a pattern P, and an integer k, identify the (starting positions of) substrings of T that are at **edit distance** of at most k to P.

Edit distance: minimum number of insertions, deletions, or substitutions of single characters to transform one string into another string



Outline

Basic Tricks and Tools, Previous Work

New Algorithms via Structural Insights

Spotlight Extension: Quantum Algorithms for ASM

Spotlight Extension: String Matching with Weighted Edits

Open Problems and Future Directions

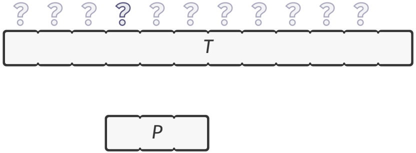
Basic Tricks and Tools: The Standard Trick

early 1980's

Approximate String Matching

Given: text T, pattern P, threshold k; Find: (starting pos. of) substrings of T at edit distance ≤ k to P.

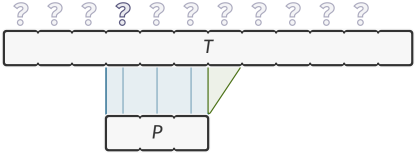
Focus: Obtain starting positions of occurrences



Approximate String Matching

early 1980's

Given: text T, pattern P, threshold k; Find: (starting pos. of) substrings of T at edit distance ≤ k to P.

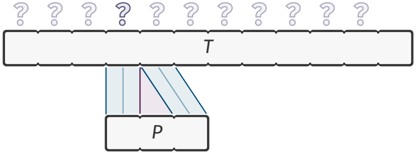


Focus: Obtain starting positions of occurrences

Approximate String Matching

early 1980's

Given: text T, pattern P, threshold k; Find: (starting pos. of) substrings of T at edit distance $\leq k$ to P.



Focus: Obtain starting positions of occurrences

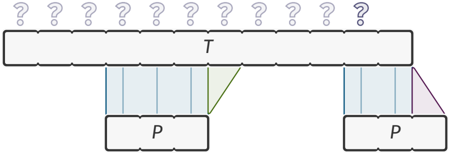
Basic Tricks and Tools: The Standard Trick

early 1980's

Approximate String Matching

Given: text T, pattern P, threshold k; Find: (starting pos. of) substrings of T at edit distance $\leq k$ to P.

Focus: Obtain starting positions of occurrences



Basic Tricks and Tools: The Standard Trick

early 1980's

Approximate String Matching

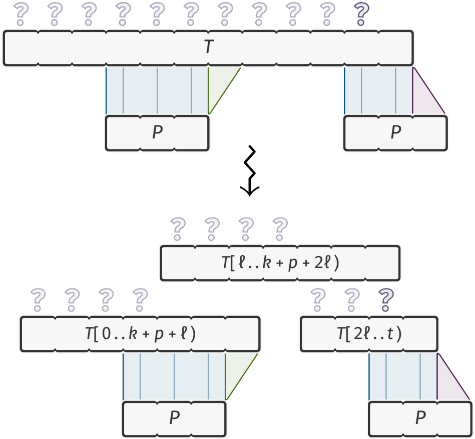
Given: text T, pattern P, threshold k; Find: (starting pos. of) substrings of T at edit distance $\leq k$ to P.

Focus: Obtain starting positions of occurrences

"Standard Trick": write $t := |T|$, $p := |P|$

Split T into overlapping fragments of len $\ell + p + k$

$\rightsquigarrow$ $O(t/\ell)$ instances,
each "responsible" for its first ℓ positions



Basic Tricks and Tools: The Standard Trick

early 1980's

Approximate String Matching

Given: text T, pattern P, threshold k; Find: (starting pos. of) substrings of T at edit distance $\leq k$ to P.

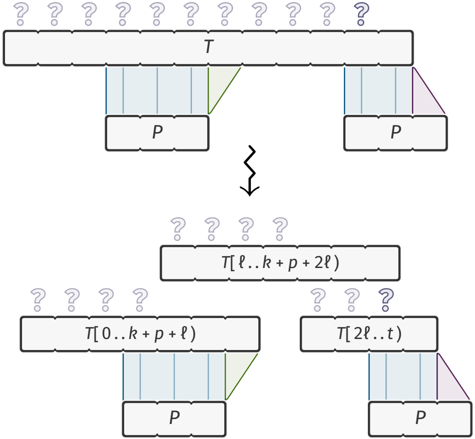
Focus: Obtain starting positions of occurrences

"Standard Trick": write $t := |T|$, $p := |P|$

Split T into overlapping fragments of len $\ell + p + k$

$\rightsquigarrow$ $O(t/\ell)$ instances,
each "responsible" for its first ℓ positions

$\rightsquigarrow$ Useful special cases: $\ell = k$ and $\ell = 0.5\,p$

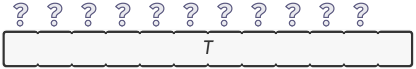


Basic Tricks and Tools: (Filter and) Verify

early 1980's

Approximate String Matching

Given: text T, pattern P, integer k; Find: (starting pos. of) substrings of T at edit distance ≤ k to P.



Focus: Obtain starting positions of occurrences

"Filter and Verify Paradigm"

Basic Tricks and Tools: (Filter and) Verify

early 1980's

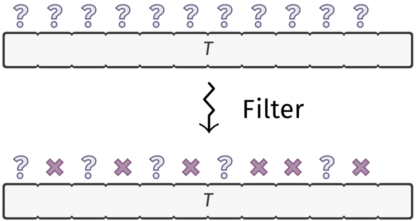
Approximate String Matching

Given: text T, pattern P, integer k; Find: (starting pos. of) substrings of T at edit distance ≤ k to P.

Focus: Obtain starting positions of occurrences

“Filter and Verify Paradigm”

Step 1, Filter: (typically fast)
Compute (small) superset of starting positions



Basic Tricks and Tools: (Filter and) Verify

Approximate String Matching

early 1980's

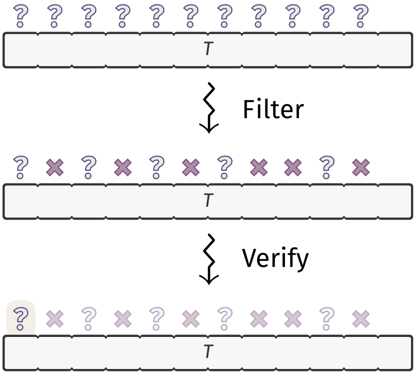
Given: text T, pattern P, integer k; Find: (starting pos. of) substrings of T at edit distance ≤ k to P.

Focus: Obtain starting positions of occurrences

"Filter and Verify Paradigm"

Step 1, Filter: (typically fast)
Compute (small) superset of starting positions

Step 2, Verify: (typically slow)
Check for occ at each remaining position



Basic Tricks and Tools: (Filter and) Verify

Approximate String Matching

early 1980's

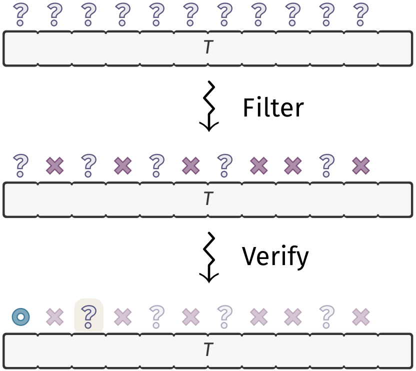
Given: text T, pattern P, integer k; Find: (starting pos. of) substrings of T at edit distance ≤ k to P.

Focus: Obtain starting positions of occurrences

"Filter and Verify Paradigm"

Step 1, Filter: (typically fast)
Compute (small) superset of starting positions

Step 2, Verify: (typically slow)
Check for occ at each remaining position



Basic Tricks and Tools: (Filter and) Verify

Approximate String Matching

early 1980's

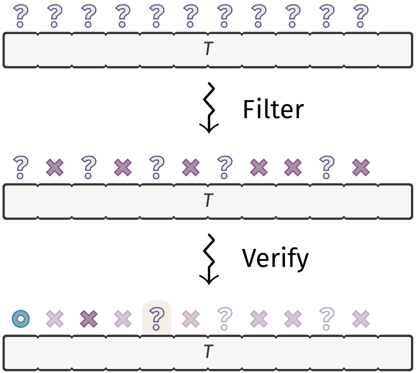
Given: text T, pattern P, integer k; Find: (starting pos. of) substrings of T at edit distance $\le k$ to P.

Focus: Obtain starting positions of occurrences

"Filter and Verify Paradigm"

Step 1, Filter: (typically fast)
Compute (small) superset of starting positions

Step 2, Verify: (typically slow)
Check for occ at each remaining position



Basic Tricks and Tools: (Filter and) Verify

Approximate String Matching

early 1980's

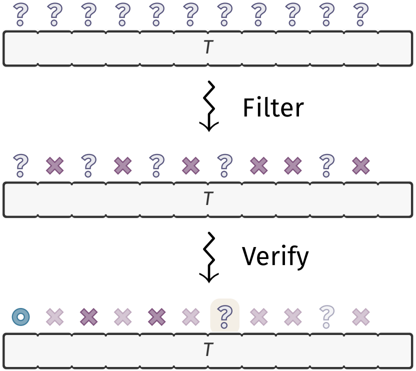
Given: text T, pattern P, integer k; Find: (starting pos. of) substrings of T at edit distance ≤ k to P.

Focus: Obtain starting positions of occurrences

"Filter and Verify Paradigm"

Step 1, Filter: (typically fast)
Compute (small) superset of starting positions

Step 2, Verify: (typically slow)
Check for occ at each remaining position



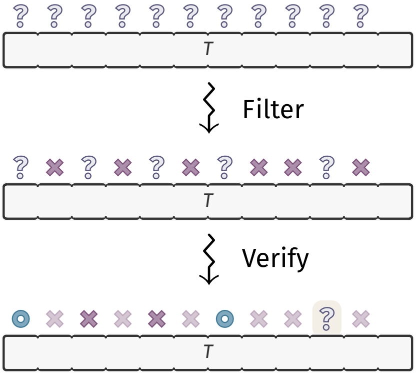
Basic Tricks and Tools: (Filter and) Verify

Approximate String Matching

early 1980's

Given: text T, pattern P, integer k;   Find: (starting pos. of) substrings of T at edit distance ≤ k to P.

Focus: Obtain starting positions of occurrences



"Filter and Verify Paradigm"

Step 1, Filter: (typically fast)
Compute (small) superset of starting positions

Step 2, Verify: (typically slow)
Check for occ at each remaining position

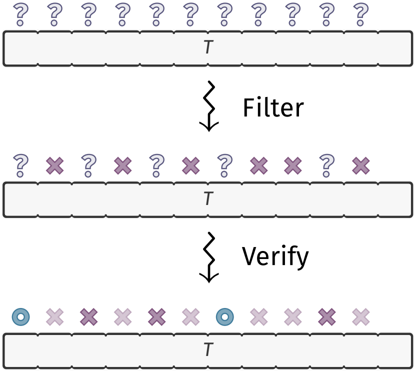
Basic Tricks and Tools: (Filter and) Verify

early 1980's

Approximate String Matching

Given: text T, pattern P, integer k; Find: (starting pos. of) substrings of T at edit distance ≤ k to P.

Focus: Obtain starting positions of occurrences



"Filter and Verify Paradigm"

Step 1, Filter: (typically fast)
Compute (small) superset of starting positions

Step 2, Verify: (typically slow)
Check for occ at each remaining position

Basic Tricks and Tools: (Filter and) Verify

Approximate String Matching

early 1980's

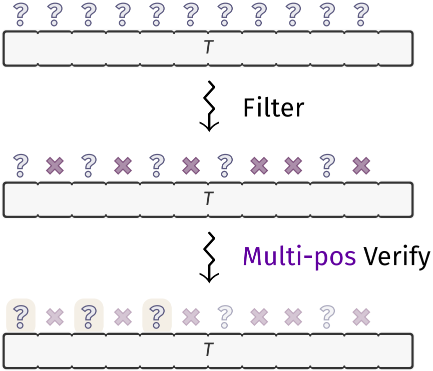
Given: text T, pattern P, integer k; Find: (starting pos. of) substrings of T at edit distance ≤ k to P.

Focus: Obtain starting positions of occurrences

"Filter and Verify Paradigm"

Step 1, Filter: (typically fast)
Compute (small) superset of starting positions

Step 2', Multi-pos Verify: (typically slow)
Check for occ at each remaining position,
multiple positions at once



Basic Tricks and Tools: (Filter and) Verify

Approximate String Matching

early 1980's

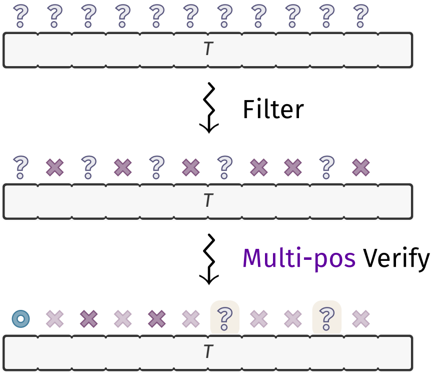
Given: text T, pattern P, integer k; Find: (starting pos. of) substrings of T at edit distance ≤ k to P.

Focus: Obtain starting positions of occurrences

"Filter and Verify Paradigm"

Step 1, Filter: (typically fast)
Compute (small) superset of starting positions

Step 2', Multi-pos Verify: (typically slow)
Check for occ at each remaining position,
multiple positions at once



Basic Tricks and Tools: (Filter and) Verify

Approximate String Matching

early 1980's

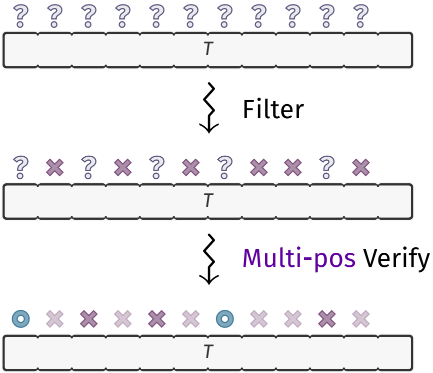
Given: text T, pattern P, integer k; Find: (starting pos. of) substrings of T at edit distance ≤ k to P.

Focus: Obtain starting positions of occurrences

"Filter and Verify Paradigm"

Step 1, Filter: (typically fast)
Compute (small) superset of starting positions

Step 2', Multi-pos Verify: (typically slow)
Check for occ at each remaining position, multiple positions at once

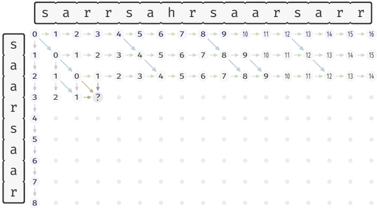


early 1980's

Approximate String Matching

Given: text T, pattern P, threshold k; Find: (starting pos. of) substrings of T at edit distance $\leq k$ to P.

Edit dist/Verify Algorithm	ASM Algorithm	($t := \lvert T \rvert, p := \lvert P \rvert$)
Textbook DP ⇝ $O(tp)$ [Sellers, 1980] and others	Verify pos 1 by 1 ⇝ $O(t^2p)$	substr



$e_{i,0} = i$ (delete $T[0..i)$)

$e_{0,j} = j$ (insert $P[0..j)$)

$$e_{i,j} = \min \begin{cases} e_{i-1,j} + 1 & \text{(del from } T\text{)} \\ e_{i,j-1} + 1 & \text{(ins in } T\text{)} \\ e_{i-1,j-1} + [T[i] \neq P[j]] & \text{(match/subst)} \end{cases}$$

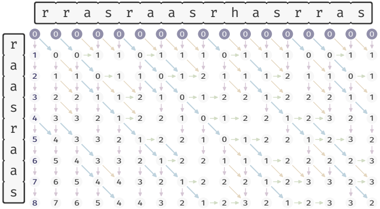
Classical Algorithms

Approximate String Matching

early 1980's

Given: text T, pattern P, threshold k; Find: (starting pos. of) substrings of T at edit distance $\leq k$ to P.

Edit dist/Verify Algorithm	ASM Algorithm	$(t := \|T\|, p := \|P\|)$
Textbook DP $\rightsquigarrow$ $O(tp)$ [Sellers, 1980] and others	Verify pos 1 by 1 $\rightsquigarrow$ $O(t^2p)$ Verify all pos at once $\rightsquigarrow$ $O(tp)$	substr start pos



		r	r	a	s	r	a	a	s	r	h	a	s	r	r	a	s
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
r	1	0	0	1	1	0	1	1	1	0	1	1	1	0	0	1	1
a	2	1	1	0	1	1	0	1	2	1	1	1	2	1	1	0	1
a	3	2	2	1	1	2	1	0	1	2	2	1	2	2	2	1	1
s	4	3	3	2	1	2	2	1	0	1	2	2	1	2	3	2	1
r	5	4	3	3	2	1	2	2	1	0	1	2	2	1	2	3	2
a	6	5	4	3	3	2	1	2	2	1	1	1	2	2	2	2	3
a	7	6	5	4	4	3	2	1	2	2	2	1	2	3	3	2	3
s	8	7	6	5	4	4	3	2	1	2	3	2	1	2	3	3	2

$e_{i,0} = 0$ (delete $T[\,0\,..\,i\,)$)

$e_{0,j} = j$ (insert $P[\,0\,..\,j\,)$)

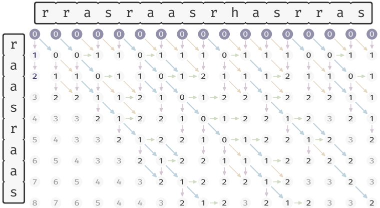
$$e_{i,j} = \min \begin{cases} e_{i-1,j} + 1 & \text{(del from } T\text{)} \\ e_{i,j-1} + 1 & \text{(ins in } T\text{)} \\ e_{i-1,j-1} + [T[\,i\,] \neq P[\,j\,]] & \text{(match/subst)} \end{cases}$$

Approximate String Matching

early 1980's

Given: text T, pattern P, threshold k; Find: (starting pos. of) substrings of T at edit distance $\leq k$ to P.

Edit dist/Verify Algorithm	ASM Algorithm	$(t := \|T\|, p := \|P\|)$
Textbook DP $\rightsquigarrow O(tp)$ [Sellers, 1980] and others	Verify pos 1 by 1 $\rightsquigarrow O(t^2p)$ Verify all pos at once $\rightsquigarrow O(tp)$	substr start pos
DP, diagonally compute only entries $\leq k$ $\rightsquigarrow O(t + kp)$	Verify pos 1 by 1 $\rightsquigarrow O(tkp)$ Verify $\ell \leq t$ pos at once $\rightsquigarrow O(t/\ell \cdot (\ell + k)p) = O(tp)$	substr start pos



		r	r	a	s	r	a	a	s	r	h	a	s	r	r	a	s
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
r	1	0	0	1	1	0	1	1	1	0	1	1	1	0	0	1	1
a	2	1	1	0	1	1	0	1	2	1	1	1	2	1	1	0	1
a	3	2	2	1	1	2	1	0	1	2	2	1	2	2	2	1	1
s	4	3	3	2	1	2	2	1	0	1	2	2	1	2	3	2	1
r	5	4	3	3	2	1	2	2	1	0	1	2	2	1	2	3	2
a	6	5	4	3	3	2	1	2	2	1	1	1	2	2	2	2	3
a	7	6	5	4	4	3	2	1	2	2	2	1	2	3	3	2	3
s	8	7	6	5	4	4	3	2	1	2	3	2	1	2	3	3	2

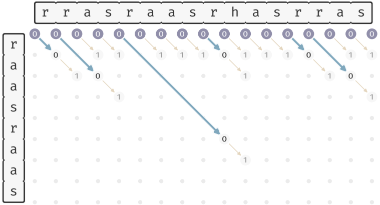
Prune whenever $e_{i,j} > k$.

Approximate String Matching

early 1980's

Given: text T, pattern P, threshold k; Find: (starting pos. of) substrings of T at edit distance $\leq k$ to P.

Edit dist/Verify Algorithm	ASM Algorithm	$(t := \|T\|, p := \|P\|)$
Textbook DP $\rightsquigarrow O(tp)$ [Sellers, 1980] and others	Verify pos 1 by 1 $\rightsquigarrow O(t^2p)$ Verify all pos at once $\rightsquigarrow O(tp)$	substr start pos
DP, diagonally compute only entries $\leq k$ $\rightsquigarrow O(t + kp)$	Verify pos 1 by 1 $\rightsquigarrow O(tkp)$ Verify $\ell \leq t$ pos at once $\rightsquigarrow O(t/\ell \cdot (\ell + k)p) = O(tp)$	substr start pos
compute DP diagonals, jump over common substr in $O(1)$ $\rightsquigarrow O(t + k^2)$ [Landau, Vishkin'89].	Verify pos 1 by 1 $\rightsquigarrow O(tk^2)$ Verify $\ell \leq t$ pos at once $\rightsquigarrow O(t/\ell \cdot (\ell + k)k) = O(tk)$	substr start pos



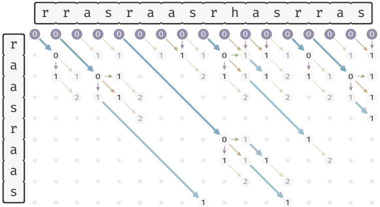
Compute furthest pos on diag d reachable w/ $\ell = 0, \dots, k$ edits.
$\rightsquigarrow$ Jump over equal substrings in $O(1)$ time

Approximate String Matching

early 1980's

Given: text T, pattern P, threshold k; Find: (starting pos. of) substrings of T at edit distance $\leq k$ to P.

Edit dist/Verify Algorithm	ASM Algorithm	$(t := \|T\|, p := \|P\|)$
Textbook DP $\rightsquigarrow O(tp)$ [Sellers, 1980] and others	Verify pos 1 by 1 $\rightsquigarrow O(t^2p)$ Verify all pos at once $\rightsquigarrow O(tp)$	substr start pos
DP, diagonally compute only entries $\leq k$ $\rightsquigarrow O(t + kp)$	Verify pos 1 by 1 $\rightsquigarrow O(tkp)$ Verify $\ell \leq t$ pos at once $\rightsquigarrow O(t/\ell \cdot (\ell + k)p) = O(tp)$	substr start pos
compute DP diagonals, jump over common substr in $O(1)$ $\rightsquigarrow O(t + k^2)$ [Landau, Vishkin'89].	Verify pos 1 by 1 $\rightsquigarrow O(tk^2)$ Verify $\ell \leq t$ pos at once $\rightsquigarrow O(t/\ell \cdot (\ell + k)k) = O(tk)$	substr start pos



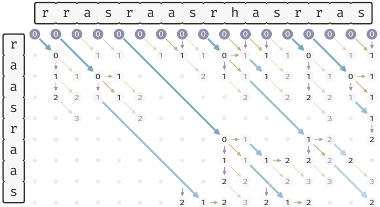
Compute furthest pos on diag d reachable w/ $\ell = 0, \ldots, k$ edits.
$\rightsquigarrow$ Jump over equal substrings in $O(1)$ time

early 1980's

Approximate String Matching

Given: text T, pattern P, threshold k; Find: (starting pos. of) substrings of T at edit distance $\leq k$ to P.

Edit dist/Verify Algorithm	ASM Algorithm	$(t := \|T\|, p := \|P\|)$
Textbook DP $\rightsquigarrow O(tp)$ [Sellers, 1980] and others	Verify pos 1 by 1 $\rightsquigarrow O(t^2p)$ Verify all pos at once $\rightsquigarrow O(tp)$	substr start pos
DP, diagonally compute only entries $\leq k$ $\rightsquigarrow O(t + kp)$	Verify pos 1 by 1 $\rightsquigarrow O(tkp)$ Verify $\ell \leq t$ pos at once $\rightsquigarrow O(t/\ell \cdot (\ell + k)p) = O(tp)$	substr start pos
compute DP diagonals, jump over common substr in $O(1)$ $\rightsquigarrow O(t + k^2)$ [Landau, Vishkin'89].	Verify pos 1 by 1 $\rightsquigarrow O(tk^2)$ Verify $\ell \leq t$ pos at once $\rightsquigarrow O(t/\ell \cdot (\ell + k)k) = O(tk)$	substr start pos



Compute furthest pos on diag d reachable w/ $\ell = 0, \ldots, k$ edits.
$\rightsquigarrow$ Jump over equal substrings in $O(1)$ time

early 1980's

Approximate String Matching

Given: text T, pattern P, threshold k; Find: (starting pos. of) substrings of T at edit distance $\leq k$ to P.

Edit dist/Verify Algorithm	ASM Algorithm	$(t := \lvert T\rvert, p := \lvert P\rvert)$
Textbook DP ⇝ $O(tp)$ [Sellers, 1980] and others	Verify pos 1 by 1 ⇝ $O(t^2p)$ Verify all pos at once ⇝ $O(tp)$	substr start pos
DP, diagonally compute only entries $\leq k$ ⇝ $O(t + kp)$	Verify pos 1 by 1 ⇝ $O(tkp)$ Verify $\ell \leq t$ pos at once ⇝$O(t/\ell \cdot (\ell + k)p) = O(tp)$	substr start pos
compute DP diagonals, jump over common substr in $O(1)$ ⇝ $O(t + k^2)$ [Landau, Vishkin'89].	Verify pos 1 by 1 ⇝ $O(tk^2)$ Verify $\ell \leq t$ pos at once ⇝ $O(t/\ell \cdot (\ell + k)k) = O(tk)$	substr start pos

⇝ Didn't use filter yet...

Classical Algorithms

early 1980's

Approximate String Matching

Given: text T, pattern P, integer k; Find: (starting pos. of) substrings of T at edit distance $\leq k$ to P.

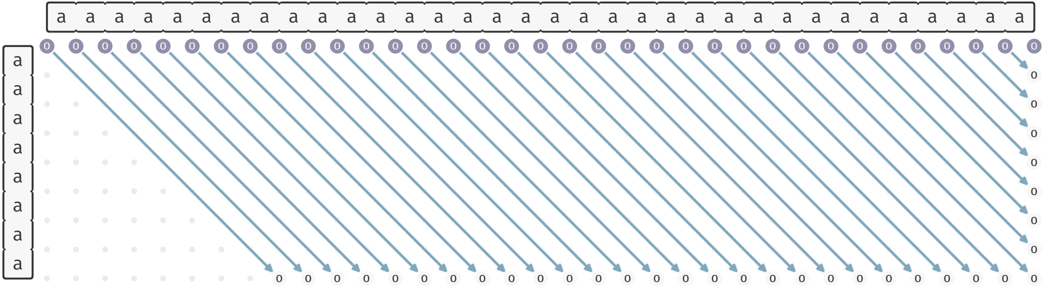
Idea: Can we filter out diagonals that will never lead to an occurrence?

early 1980's

Approximate String Matching

Given: text T, pattern P, integer k; Find: (starting pos. of) substrings of T at edit distance ≤ k to P.

Idea: Can we filter out diagonals that will never lead to an occurrence?



Classical Algorithms

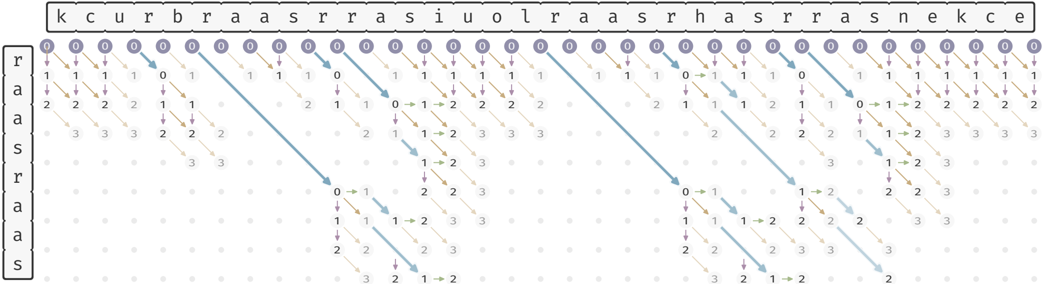
Approximate String Matching

early 1980's

Given: text *T*, pattern *P*, integer *k*; Find: (starting pos. of) substrings of *T* at edit distance ≤ *k* to *P*.

Idea: Can we filter out diagonals that will never lead to an occurrence?

Sometimes we can; and if we cannot, there might be structure to exploit!


k c u r b r a a s r r a s i u o l r a a s r h a s r r a s n e k c e
r a a s r a a s

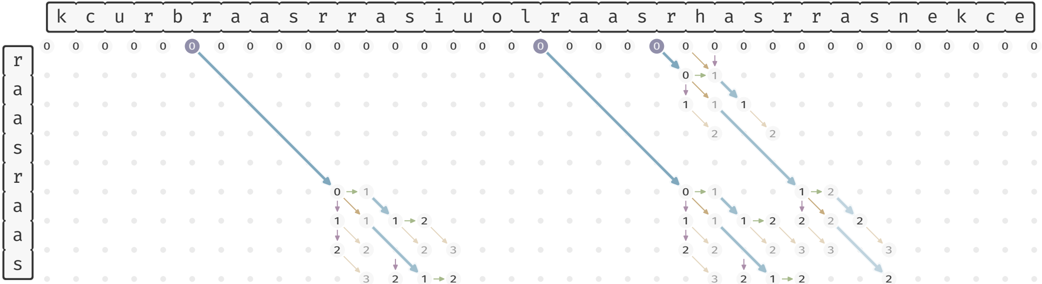
early 1980's

Approximate String Matching

Given: text *T*, pattern *P*, integer *k*; Find: (starting pos. of) substrings of *T* at edit distance ≤ *k* to *P*.

Idea: Can we filter out diagonals that will never lead to an occurrence?

Sometimes we can; and if we cannot, there might be structure to exploit!


k c u r b r a a s r r a s i u o l r a a s r h a s r r a s n e k c e
r a a s r a a s

early 1980's

Approximate String Matching

Given: text T, pattern P, integer k; Find: (starting pos. of) substrings of T at edit distance $\leq k$ to P.

- Idea: Find parts of P that are rare in T

  ⇝ Can filter out candidates for occurrences

Classical Algorithms

Approximate String Matching

early 1980's

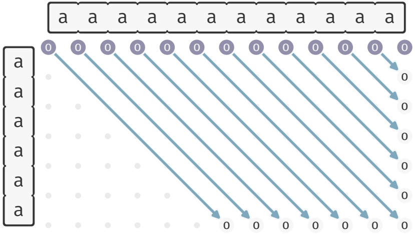
Given: text *T*, pattern *P*, integer *k*; Find: (starting pos. of) substrings of *T* at edit distance ≤ *k* to *P*.

- Idea: Find parts of *P* that are rare in *T*

  ⇝ Can filter out candidates for occurrences

- Seen: Highly repetitive parts *R* are bad...

  ⇝ *R* may appear ≈ *t* times in *T*

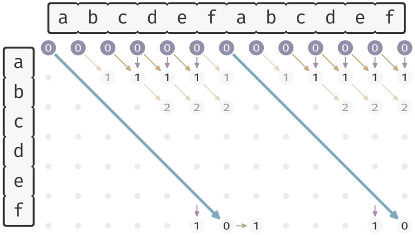


early 1980's

Approximate String Matching

Given: text T, pattern P, integer k; Find: (starting pos. of) substrings of T at edit distance $\leq k$ to P.

- Idea: Find parts of P that are rare in T
  - ⇝ Can filter out candidates for occurrences
- Seen: Highly repetitive parts R are bad...
  - ⇝ R may appear $\approx t$ times in T
- Non-repetitive parts B are good!
  - ⇝ B may appear only $\approx t/b$ times in T

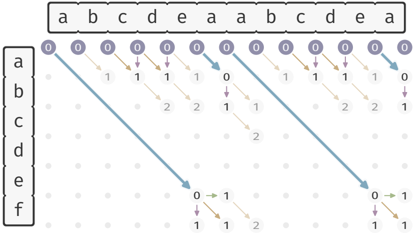


Approximate String Matching

early 1980's

Given: text *T*, pattern *P*, integer *k*; Find: (starting pos. of) substrings of *T* at edit distance ≤ *k* to *P*.

- Idea: Find parts of *P* that are rare in *T*
  - ⇝ Can filter out candidates for occurrences
- Seen: Highly repetitive parts *R* are bad...
  - ⇝ *R* may appear ≈ *t* times in *T*
- Non-repetitive parts *B* are good!
  - ⇝ *B* may appear only ≈ *t*/*b* times in *T*
  - ⇝ Problem: *B* may appear approximately in *T*

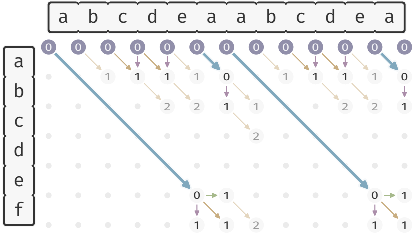


Approximate String Matching

early 1980's

Given: text T, pattern P, integer k; Find: (starting pos. of) substrings of T at edit distance ≤ k to P.

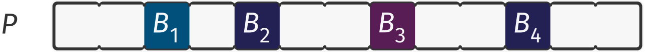
- Idea: Find parts of P that are rare in T
  - ⇝ Can filter out candidates for occurrences
- Seen: Highly repetitive parts R are bad...
  - ⇝ R may appear ≈ t times in T
- Non-repetitive parts B are good!
  - ⇝ B may appear only ≈ t/b times in T
  - ⇝ Problem: B may appear approximately in T
  - ⇝ Use > k disjoint breaks



Classical Algorithms: More on [Cole, Hariharan'98]

Suppose we have: $2k$ disjoint breaks $B_1, \dots, B_{2k}$ in P such that

- $|B_i| = \Theta(p/k)$ and
- B_i is not periodic (no long prefix is also a suffix)

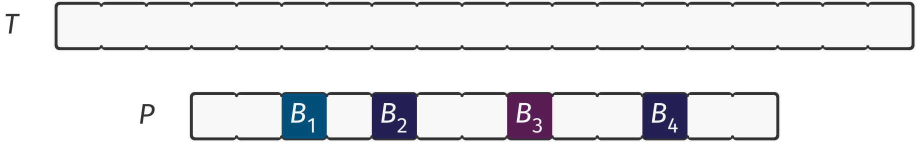


Classical Algorithms: More on [Cole, Hariharan'98]

Suppose we have: $2k$ disjoint breaks $B_1, \dots, B_{2k}$ in P such that

- $|B_i| = \Theta(p/k)$ and
- B_i is not periodic (no long prefix is also a suffix)

- In any k-edit occ, at least 1 break is matched exactly. (budget for edits is k)

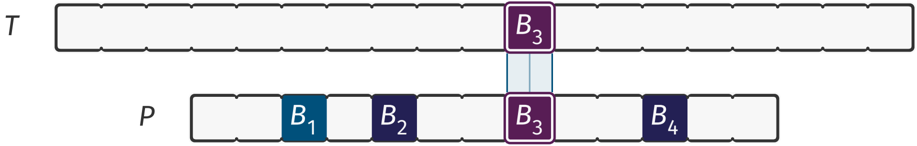


Classical Algorithms: More on [Cole, Hariharan'98]

Suppose we have: $2k$ disjoint breaks $B_1, \dots, B_{2k}$ in P such that

- $|B_i| = \Theta(p/k)$ and
- B_i is not periodic (no long prefix is also a suffix)

- In any k-edit occ, at least 1 break is matched exactly. (budget for edits is k)
- Algorithm idea: For each B_i, find exact matches of B_i in T

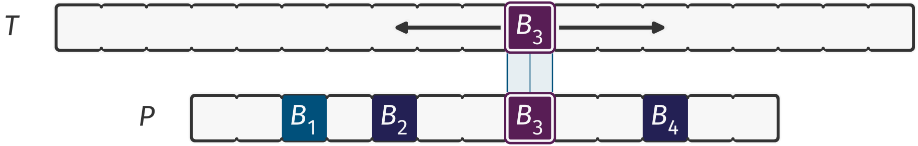


Classical Algorithms: More on [Cole, Hariharan'98]

Suppose we have: $2k$ disjoint breaks $B_1, \dots, B_{2k}$ in P such that

- $|B_i| = \Theta(p/k)$ and
- B_i is not periodic (no long prefix is also a suffix)

- In any k-edit occ, at least 1 break is matched exactly. (budget for edits is k)
- Algorithm idea: For each B_i, find exact matches of B_i in T,
  Try to extend each exact match into an occ using $O(n + k^2)$-time Verify [LV'89]

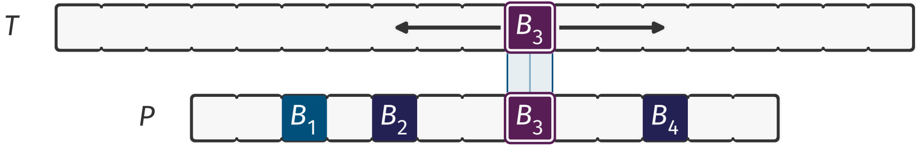


Classical Algorithms: More on [Cole, Hariharan'98]

Suppose we have: $2k$ disjoint breaks $B_1, \dots, B_{2k}$ in P such that

- $|B_i| = \Theta(p/k)$ and
- B_i is not periodic (no long prefix is also a suffix)

- In any k-edit occ, at least 1 break is matched exactly. (budget for edits is k)
- Algorithm idea: For each B_i, find exact matches of B_i in T,
  Try to extend each exact match into an occ using $O(n + k^2)$-time Verify [LV'89]
- Have at most $t/(|B_i|/2) = \Theta(kt/p)$ exact occ's of B_i

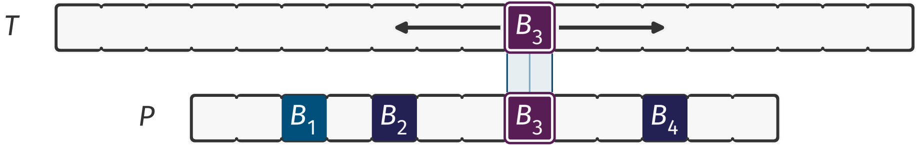


Classical Algorithms: More on [Cole, Hariharan'98]

Suppose we have: $2k$ disjoint breaks $B_1, \dots, B_{2k}$ in P such that

- $|B_i| = \Theta(p/k)$ and
- B_i is not periodic (no long prefix is also a suffix)

- In any k-edit occ, at least 1 break is matched exactly. (budget for edits is k)
- Algorithm idea: For each B_i, find exact matches of B_i in T,
  Try to extend each exact match into an occ using $O(n + k^2)$-time Verify [LV'89]
- Have at most $t/(|B_i|/2) = \Theta(kt/p)$ exact occ's of B_i
  $\rightsquigarrow O(k^2 t/p)$ calls to [LV'89]; $O(k^4 t/p)$ time in total

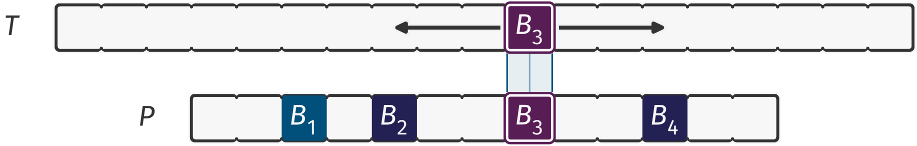


Classical Algorithms: More on [Cole, Hariharan'98]

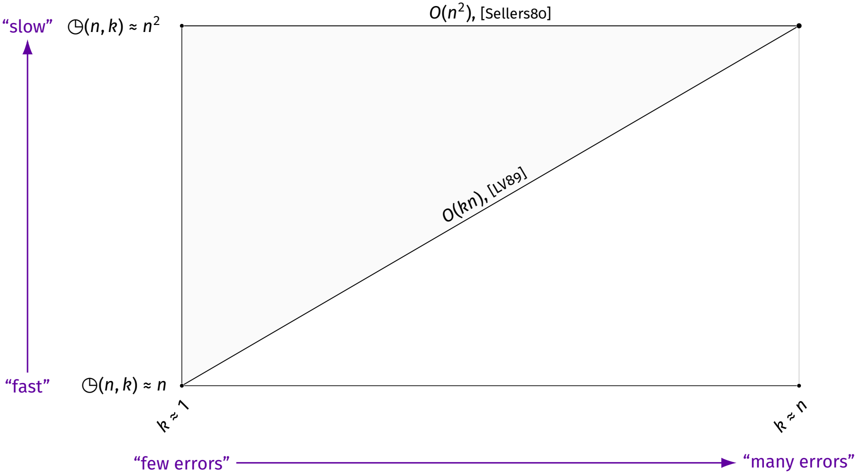
Suppose we have: $2k$ disjoint breaks $B_1, \dots, B_{2k}$ in P such that

- $|B_i| = \Theta(p/k)$ and
- B_i is not periodic (no long prefix is also a suffix)

- In any k-edit occ, at least k breaks are matched exactly. (budget for edits is k)
- Algorithm idea: For each B_i, find exact matches of B_i in T, Try to extend each exact match into an occ using $O(n + k^2)$-time Verify [LV'89]
- Have at most $t/(|B_i|/2) = \Theta(kt/p)$ exact occ's of B_i $\rightsquigarrow$ $O(k^2 t/p)$ calls to [LV'89]; $O(k^4 t/p)$ time in total
- Can even be improved to $O(k^3 t/p)$; but not-enough breaks case is also $O(k^4 t/p)$.

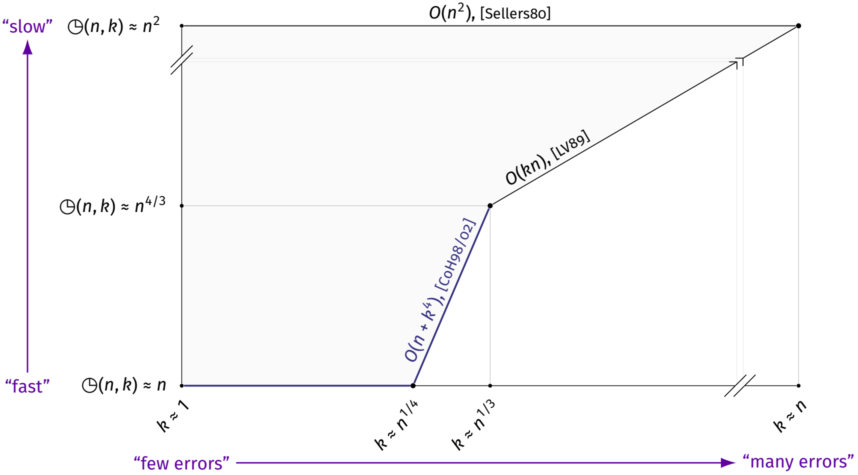


Recap: State-of-the-Art Algorithms



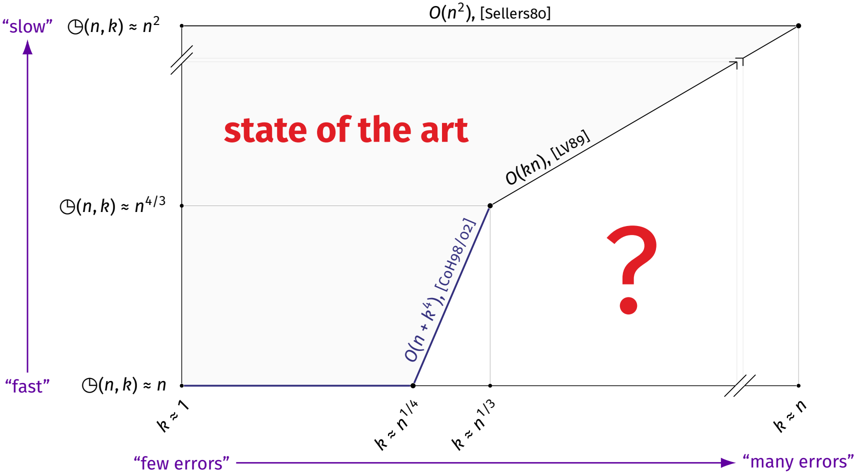
Existing algorithms for the case $n \approx t \approx 2p$

Recap: State-of-the-Art Algorithms



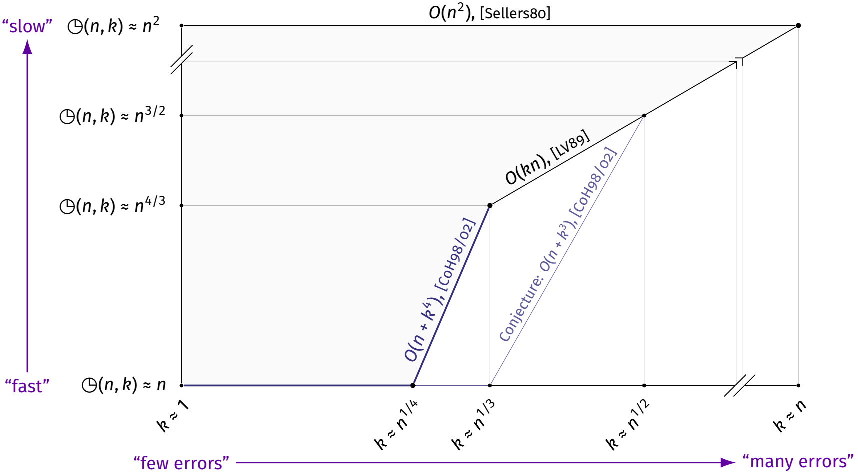
Existing algorithms for the case $n \approx t \approx 2p$

Recap: State-of-the-Art Algorithms



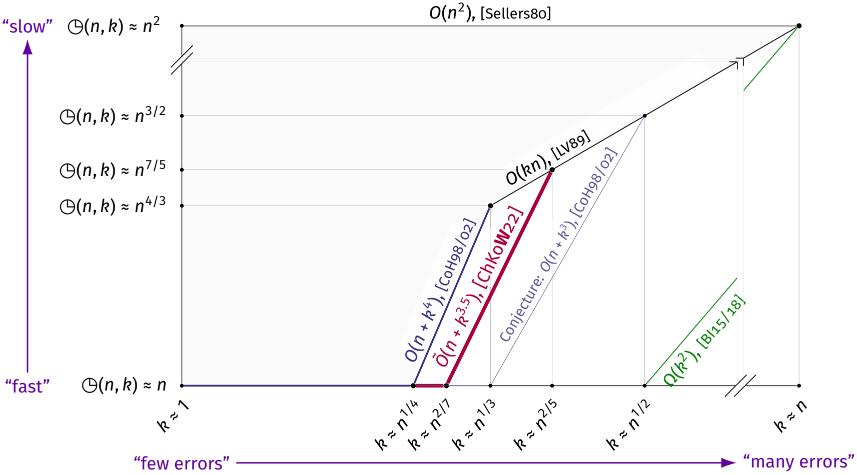
Existing algorithms for the case $n \approx t \approx 2p$

Recap: State-of-the-Art Algorithms



Existing algorithms for the case $n \approx t \approx 2p$

Recap: State-of-the-Art Algorithms



Existing algorithms for the case $n \approx t \approx 2p$

Recap: State-of-the-Art Algorithms

Why did a new improvement take 24 years?

Detour: Why did a new improvement for Approximate String Matching take 24 years?

- Simpler problems were not fully understood!

Detour: Why did a new improvement for Approximate String Matching take 24 years?

- Simpler problems were not fully understood!
- Example 1: The complexity of Edit Distance was settled only in 2014

Hardness of Edit Distance [BI18] (ann. 2014)

Computing the Edit Distance of to strings of length n is not possible in $O(n^{2-\varepsilon})$ time.

(Unless there is a major breakthrough for the Satisfiability Problem.)

Detour: Why did a new improvement for Approximate String Matching take 24 years?

- Simpler problems were not fully understood!
- Example 1: The complexity of Edit Distance was settled only in 2014

[BI18] (ann. 2014)

Hardness of Edit Distance

Computing the Edit Distance of to strings of length n is not possible in $O(n^{2-\varepsilon})$ time.

(Unless there is a major breakthrough for the Satisfiability Problem.)

⇝ Yields lower bound of $O(t + k^2 \cdot t/p)$ for Approximate String Matching.

Detour: Why did a new improvement for Approximate String Matching take 24 years?

- Simpler problems were not fully understood!
- Example 2: String Matching with Mismatches (no insertions or deletions of chars)

Detour: Why did a new improvement for Approximate String Matching take 24 years?

- Simpler problems were not fully understood!
- Example 2: String Matching with Mismatches (no insertions or deletions of chars)
  - $\rightsquigarrow$ $O(tk)$ algorithm [Landau, Vishkin'**86**]
  - $\rightsquigarrow$ $\tilde{O}(t\sqrt{k})$ algorithm [Amir, Lewenstein, Porat'**04**]
  - $\rightsquigarrow$ $\tilde{O}(t + t/p \cdot k^2)$ algorithm [CFPSS'**16**]

Detour: Why did a new improvement for Approximate String Matching take 24 years?

- Simpler problems were not fully understood!
- Example 2: String Matching with Mismatches (no insertions or deletions of chars)
  $\rightsquigarrow$ $O(tk)$ algorithm [Landau, Vishkin'**86**]
  $\rightsquigarrow$ $\tilde{O}(t\sqrt{k})$ algorithm [Amir, Lewenstein, Porat'**04**]
  $\rightsquigarrow$ $\tilde{O}(t + t/p \cdot k^2)$ algorithm [CFPSS'**16**]

[GU18] (ann. 2017)

Optimal String Matching with Mismatches

There is a $\tilde{O}(t + kt/\sqrt{p})$-time algorithm for String Matching with Mismatches and no significantly faster algorithm exits.

(Unless there is a major breakthrough for combinatorial Boolean Matrix Multiplication.)

Detour: Why did a new improvement for Approximate String Matching take 24 years?

- Simpler problems were not fully understood!
- Example 2: String Matching with Mismatches (no insertions or deletions of chars)
  $\rightsquigarrow$ $O(tk)$ algorithm [Landau, Vishkin'**86**]
  $\rightsquigarrow$ $\tilde{O}(t\sqrt{k})$ algorithm [Amir, Lewenstein, Porat'**04**]
  $\rightsquigarrow$ $\tilde{O}(t + t/p \cdot k^2)$ algorithm [CFPSS'**16**]

Optimal String Matching with Mismatches [GU18] (ann. 2017)

There is a $\tilde{O}(t + kt/\sqrt{p})$-time algorithm for String Matching with Mismatches and no significantly faster algorithm exits.

(Unless there is a major breakthrough for combinatorial Boolean Matrix Multiplication.)

$\rightsquigarrow$ String Matching with Mismatches was “fully” understood only in 2017.

Beating Cole and Hariharan’s Algorithm

How do we obtain faster algorithms?

Beating Cole and Hariharan’s Algorithm

How do we obtain faster algorithms?

New insights into the solution structure of
Approximate String Matching

Think: Better filter + more structure when filter fails

Step 0:

What is the solution structure of Exact String Matching?

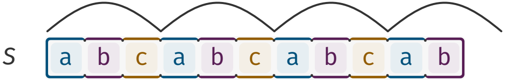
The Solution Structure of Exact String Matching

(The) Period of a String

$p > 0$ is a period of S $:\iff$ $S[i] = S[i+p]$ for all $i = 1, \dots, |S| - p$

The period of S $:\iff$ smallest period of S, write $\mathrm{per}(S)$

S is periodic $:\iff$ $\mathrm{per}(S) \le |S|/2$



$\mathrm{per}(S) = 3$

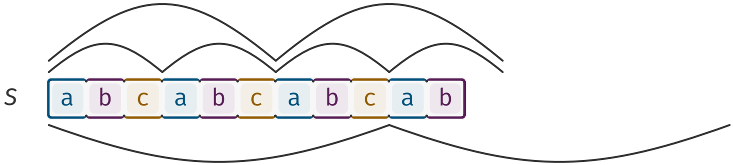
The Solution Structure of Exact String Matching

(The) Period of a String

$p > 0$ is a period of S $:\Longleftrightarrow$ $S[i] = S[i+p]$ for all $i = 1, \ldots, |S| - p$

The period of S $:\Longleftrightarrow$ smallest period of S, write $\mathrm{per}(S)$

S is periodic $:\Longleftrightarrow$ $\mathrm{per}(S) \le |S|/2$



$\mathrm{per}(S) = 3$

6 and 9 also periods of S

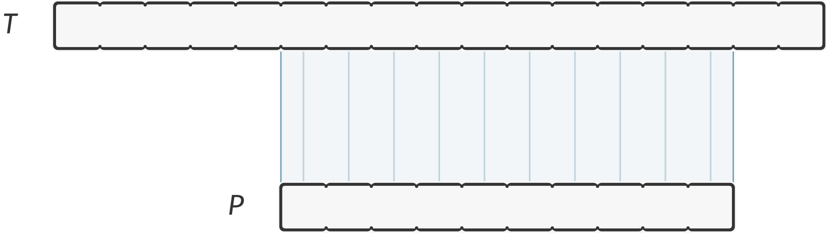
The Solution Structure of Exact String Matching

(Folklore)

"Periodicity Lemma"

Text T, pattern P with $t \leq {}^{3}\!/_{2}\, p$;   one of the following holds:

- P appears ≤ 1 times in T.



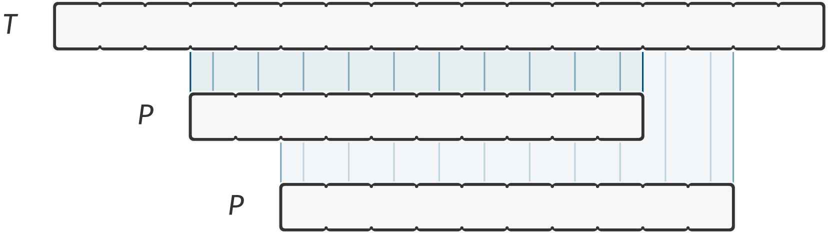
The Solution Structure of Exact String Matching

(Folklore)

"Periodicity Lemma"

Text T, pattern P with $t \leq {}^{3}\!/_{2}\, p$; one of the following holds:

- P appears ≤ 1 times in T.
- P (and the relevant part of T) are periodic with some period Q.



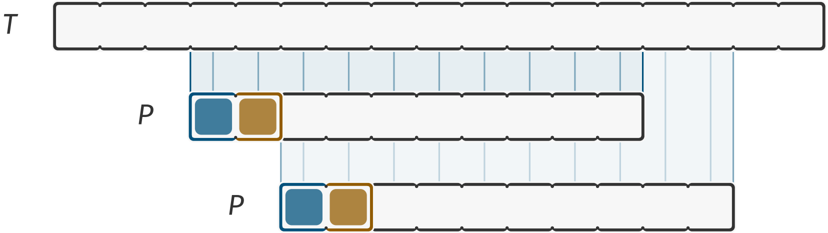
The Solution Structure of Exact String Matching

(Folklore)

“Periodicity Lemma”

Text T, pattern P with $t \le {}^{3}/_{2}\, p$; one of the following holds:

- P appears ≤ 1 times in T.
- P (and the relevant part of T) are periodic with some period Q.



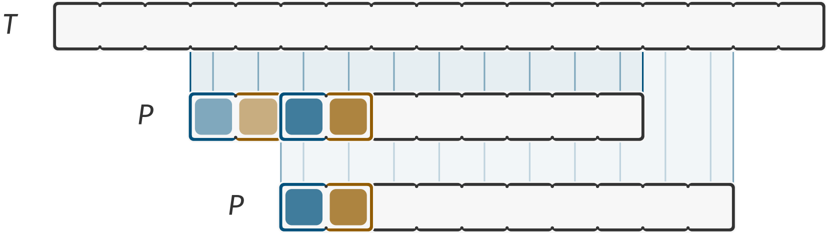
The Solution Structure of Exact String Matching

(Folklore)

“Periodicity Lemma”

Text T, pattern P with $t \leq {}^{3}/_{2}\, p$; one of the following holds:

- P appears ≤ 1 times in T.
- P (and the relevant part of T) are periodic with some period Q.



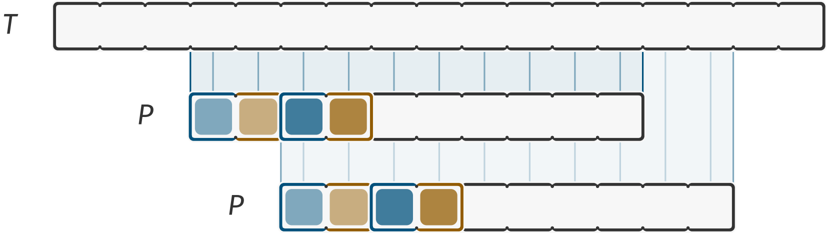
The Solution Structure of Exact String Matching

(Folklore)

“Periodicity Lemma”

Text T, pattern P with $t \leq {}^{3}\!/_{2}\, p$; one of the following holds:

- P appears ≤ 1 times in T.
- P (and the relevant part of T) are periodic with some period Q.



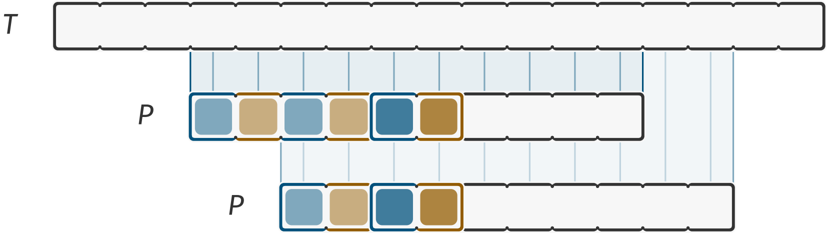
The Solution Structure of Exact String Matching

(Folklore)

"Periodicity Lemma"

Text T, pattern P with $t \leq {}^3\!/_2\, p$; one of the following holds:

- P appears ≤ 1 times in T.
- P (and the relevant part of T) are periodic with some period Q.



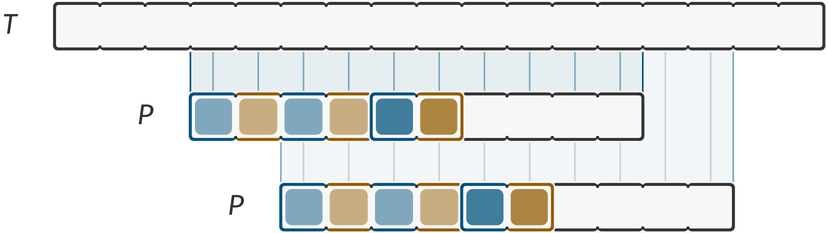
The Solution Structure of Exact String Matching

"Periodicity Lemma"

(Folklore)

Text T, pattern P with $t \leq {}^{3}\!/_{2}\, p$; one of the following holds:

- P appears ≤ 1 times in T.
- P (and the relevant part of T) are periodic with some period Q.



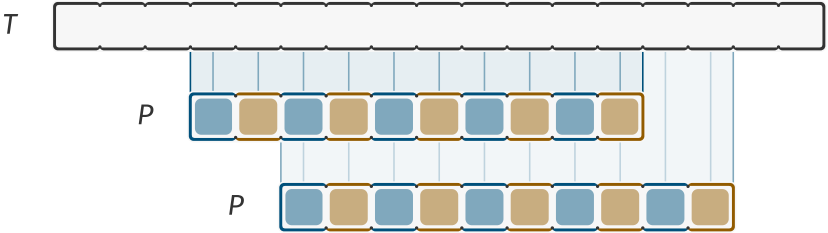
The Solution Structure of Exact String Matching

“Periodicity Lemma”

(Folklore)

Text T, pattern P with $t \le {}^{3}/_{2}\, p$; one of the following holds:

- P appears ≤ 1 times in T.
- P (and the relevant part of T) are periodic with some period Q.



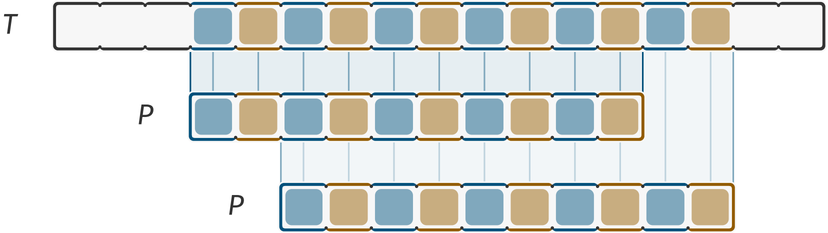
The Solution Structure of Exact String Matching

(Folklore)

"Periodicity Lemma"

Text T, pattern P with $t \leq {}^{3}/_{2}\, p$; one of the following holds:

- P appears ≤ 1 times in T.
- P (and the relevant part of T) are periodic with some period Q.



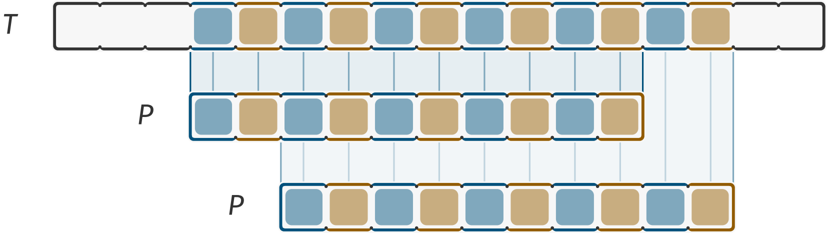
The Solution Structure of Exact String Matching

(Folklore)

“Periodicity Lemma”

Text T, pattern P with $t \leq {}^3\!/_2\, p$; one of the following holds:

- P appears ≤ 1 times in T.
- P (and the relevant part of T) are periodic with some period Q.



Standard Trick is useful here:

For $t \gg p$ consider separately $O(t/p)$ fragments of T; then Periodicity Lemma applies

Step 1:

What is the solution structure of String Matching with Mismatches?

Structural Results for Approximate String Matching

(Folklore)

Periodicity Lemma

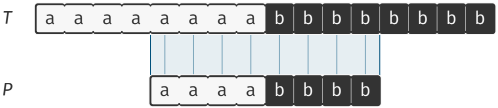
Text T, pattern P with $t \le {}^{3}/_{2}\, p$; P appears ≤ 1 times in T or P is periodic with some period Q.

[BKüW'19; ChKoW'20]

Main Result (Mismatches)

Text T, pattern P with $t \le {}^{3}/_{2}\, p$; threshold k, one of the following holds

P appears ≤ $O(k)$ times in T



P appears $2k$ times w/ ≤ k mism.

but P far from periodic

Structural Results for Approximate String Matching

Periodicity Lemma

(Folklore)

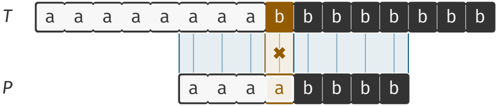
Text T, pattern P with $t \le {}^{3}/_{2}\, p$; P appears ≤ 1 times in T or P is periodic with some period Q.

Main Result (Mismatches)

[BKüW'19; ChKoW'20]

Text T, pattern P with $t \le {}^{3}/_{2}\, p$; threshold k, one of the following holds

P appears $\le O(k)$ times in T



P appears $2k$ times w/ $\le k$ mism.
but P far from periodic

Structural Results for Approximate String Matching

(Folklore)

Periodicity Lemma

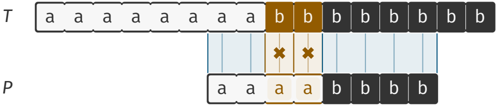
Text T, pattern P with $t \le {}^3/_2\, p$; P appears ≤ 1 times in T or P is periodic with some period Q.

[BKüW'19; ChKoW'20]

Main Result (Mismatches)

Text T, pattern P with $t \le {}^3/_2\, p$; threshold k, one of the following holds

P appears $\le O(k)$ times in T



P appears $2k$ times w/ $\le k$ mism.

but P far from periodic

Structural Results for Approximate String Matching

Periodicity Lemma

(Folklore)

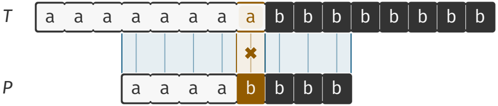
Text T, pattern P with $t \le {}^{3}/_{2}\, p$; P appears ≤ 1 times in T or P is periodic with some period Q.

Main Result (Mismatches)

[BKüW'19; ChKoW'20]

Text T, pattern P with $t \le {}^{3}/_{2}\, p$; threshold k, one of the following holds

P appears $\le O(k)$ times in T



T a a a a a a a a b b b b b b b b

P a a a a b b b b

P appears $2k$ times w/ $\le k$ mism.
but P far from periodic

Structural Results for Approximate String Matching

Periodicity Lemma

(Folklore)

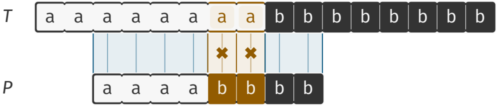
Text T, pattern P with $t \le {}^3\!/_2\, p$; P appears ≤ 1 times in T or P is periodic with some period Q.

Main Result (Mismatches)

[BKüW'19; ChKoW'20]

Text T, pattern P with $t \le {}^3\!/_2\, p$; threshold k, one of the following holds

P appears $\le O(k)$ times in T



T	a	a	a	a	a	a	a	a	b	b	b	b	b	b	b	b
							✖	✖								
P			a	a	a	a	b	b	b	b						

P appears $2k$ times w/ $\le k$ mism.

but P far from periodic

Structural Results for Approximate String Matching

Periodicity Lemma

(Folklore)

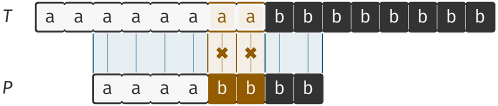
Text T, pattern P with $t \leq {}^{3}\!/_{2}\, p$; P appears ≤ 1 times in T or P is periodic with some period Q.

Main Result (Mismatches)

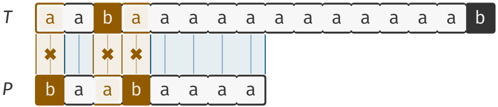
[BKüW’19; ChKoW’20]

Text T, pattern P with $t \leq {}^{3}\!/_{2}\, p$; threshold k, one of the following holds

P appears ≤ $O(k)$ times in T or P is almost periodic with some period Q



P appears $2k$ times w/ ≤ k mism.
but P far from periodic



P and T have approximate period a
(are at HD ≤ $2k$ from aa…aa)

Structural Results for Approximate String Matching

Periodicity Lemma

(Folklore)

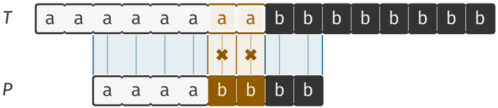
Text T, pattern P with $t \leq {}^{3}\!/_{2}\, p$; P appears ≤ 1 times in T or P is periodic with some period Q.

Main Result (Mismatches)

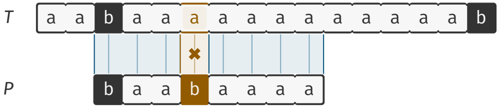
[BKüW’19; ChKoW’20]

Text T, pattern P with $t \leq {}^{3}\!/_{2}\, p$; threshold k, one of the following holds

P appears $\leq O(k)$ times in T or P is almost periodic with some period Q



P appears $2k$ times w/ $\leq k$ mism.
but P far from periodic



P and T have approximate period a
(are at HD $\leq 2k$ from aa…aa)

Structural Results for Approximate String Matching

Periodicity Lemma

(Folklore)

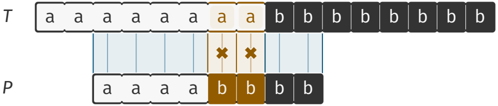
Text T, pattern P with $t \le {}^{3}\!/_{2}\, p$; P appears ≤ 1 times in T or P is periodic with some period Q.

Main Result (Mismatches)

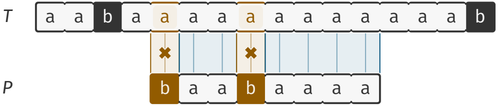
[BKüW'19; ChKoW'20]

Text T, pattern P with $t \le {}^{3}\!/_{2}\, p$; threshold k, one of the following holds

P appears $\le O(k)$ times in T or P is almost periodic with some period Q



P appears $2k$ times w/ $\le k$ mism.
but P far from periodic



P and T have approximate period a
(are at HD $\le 2k$ from aa…aa)

Structural Results for Approximate String Matching

Periodicity Lemma

(Folklore)

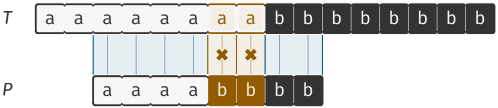
Text T, pattern P with $t \leq {}^{3}/_{2}\, p$; P appears ≤ 1 times in T or P is periodic with some period Q.

Main Result (Mismatches)

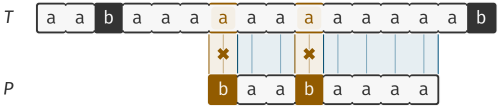
[BKüW'19; ChKoW'20]

Text T, pattern P with $t \leq {}^{3}/_{2}\, p$; threshold k, one of the following holds

P appears ≤ $O(k)$ times in T or P is almost periodic with some period Q



P appears $2k$ times w/ ≤ k mism.
but P far from periodic



P and T have approximate period a
(are at HD ≤ $2k$ from aa…aa)

Structural Results for Approximate String Matching

Periodicity Lemma

(Folklore)

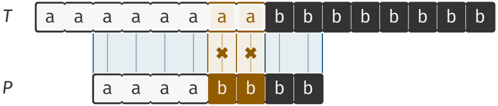
Text T, pattern P with $t \le {}^3\!/_2\, p$; P appears ≤ 1 times in T or P is periodic with some period Q.

Main Result (Mismatches)

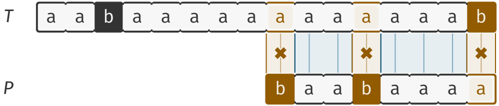
[BKüW'19; ChKoW'20]

Text T, pattern P with $t \le {}^3\!/_2\, p$; threshold k, one of the following holds

P appears ≤ $O(k)$ times in T or P is almost periodic with some period Q



P appears $2k$ times w/ ≤ k mism.
but P far from periodic



P and T have approximate period a
(are at HD ≤ $2k$ from aa…aa)

Step 1.5:

The solution structure of
Approximate String Matching.

Structural Results for Approximate String Matching

[BKüW'19; ChKoW'20]

Main Result (Mismatches)

T, P with $t \le {}^3/_2\, p$; threshold k, P appears $O(k)$ times in T or P is almost periodic with some period Q

[ChKoW'20]

Main Result (Edits)

Text T, pattern P with $t \le {}^3/_2\, p$; threshold k, one of the following holds

or P is almost periodic with some period Q

Structural Results for Approximate String Matching

Main Result (Mismatches)

[BKüW'19; ChKoW'20]

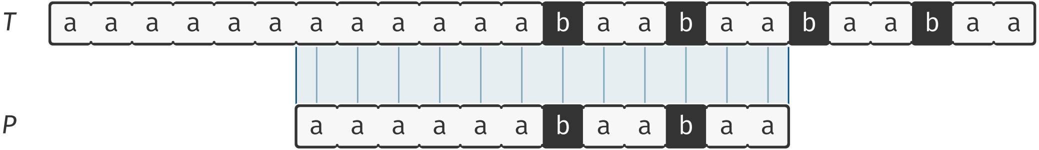
T, P with $t \le {}^3\!/_2\, p$; threshold k, P appears $O(k)$ times in T or P is almost periodic with some period Q

Main Result (Edits)

[ChKoW'20]

Text T, pattern P with $t \le {}^3\!/_2\, p$; threshold k, one of the following holds

P appears $\le O(k^2)$ times in T or P is almost periodic with some period Q



T: a a a a a a a a a a a a b a a b a a b a a b a a

P: a a a a a a b a a b a a

P appears $\approx k^2$ times with $\le k$ edits but P and T far from (approx.) periodic

Structural Results for Approximate String Matching

Main Result (Mismatches)

[BKüW'19; ChKoW'20]

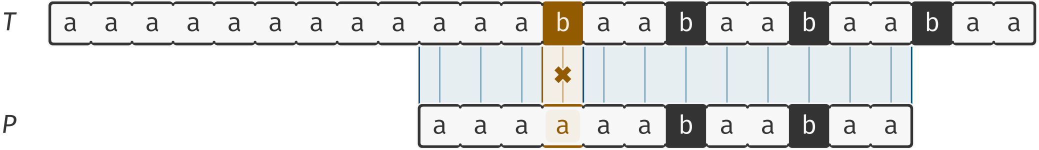
T, P with $t \le 3/2\, p$; threshold k, P appears $O(k)$ times in T or P is almost periodic with some period Q

Main Result (Edits)

[ChKoW'20]

Text T, pattern P with $t \le 3/2\, p$; threshold k, one of the following holds

P appears $\le O(k^2)$ times in T or P is almost periodic with some period Q



P appears $\approx k^2$ times with $\le k$ edits but P and T far from (approx.) periodic

Structural Results for Approximate String Matching

Main Result (Mismatches)

[BKüW'19; ChKoW'20]

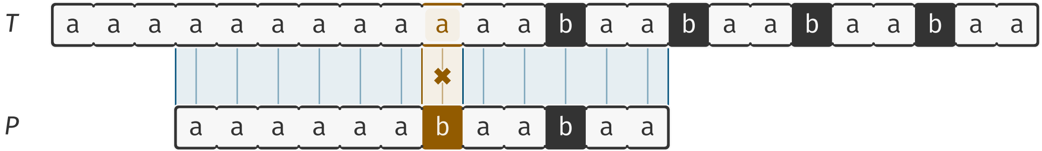
T, P with $t \le {}^3/_2\, p$; threshold k, P appears $O(k)$ times in T or P is almost periodic with some period Q

Main Result (Edits)

[ChKoW'20]

Text T, pattern P with $t \le {}^3/_2\, p$; threshold k, one of the following holds

P appears $\le O(k^2)$ times in T or P is almost periodic with some period Q



P appears $\approx k^2$ times with $\le k$ edits but P and T far from (approx.) periodic

Structural Results for Approximate String Matching

Main Result (Mismatches)

[BKüW'19; ChKoW'20]

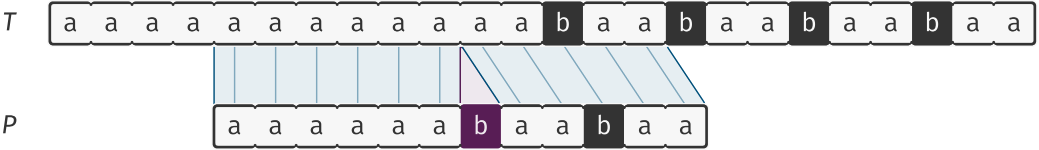
T, P with $t \leq {}^3\!/_2\, p$; threshold k, P appears $O(k)$ times in T or P is almost periodic with some period Q

Main Result (Edits)

[ChKoW'20]

Text T, pattern P with $t \leq {}^3\!/_2\, p$; threshold k, one of the following holds

P appears $\leq O(k^2)$ times in T or P is almost periodic with some period Q



T a a a a a a a a a a a a b a a b a a b a a b a a

P a a a a a a b a a b a a

P appears $\approx k^2$ times with $\leq k$ edits but P and T far from (approx.) periodic

Structural Results for Approximate String Matching

Main Result (Mismatches)

[BKüW'19; ChKoW'20]

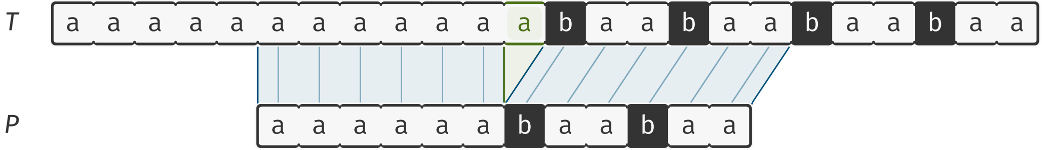
T, P with $t \leq {}^3\!/_2\, p$; threshold k, P appears $O(k)$ times in T or P is almost periodic with some period Q

Main Result (Edits)

[ChKoW'20]

Text T, pattern P with $t \leq {}^3\!/_2\, p$; threshold k, one of the following holds

P appears $\leq O(k^2)$ times in T or P is almost periodic with some period Q



T a a a a a a a a a a a a b a a b a a b a a b a a

P a a a a a a b a a b a a

P appears $\approx k^2$ times with $\leq k$ edits but P and T far from (approx.) periodic

Structural Results for Approximate String Matching

[ChKoW'20]

Main Result (Edits)

T, P with $t \leq {}^{3}\!/_{2}\, p$; threshold k, P appears $O(k^2)$ times in T or P is almost periodic with some period Q

Structural Results for Approximate String Matching

Main Result (Edits)

[ChKoW'20]

T, P with $t \leq {}^3/_2\, p$; threshold k, P appears $O(k^2)$ times in T or P is almost periodic with some period Q

Key Intermediate Result (Analyze)

[ChKoW'20]

Every string P satisfies at least one of

- P is almost periodic. aaacaaaaaaaaacaaaaaaa

Structural Results for Approximate String Matching

[ChKoW’20]

Main Result (Edits)

T, P with $t \leq {}^3\!/_2\, p$; threshold k, P appears $O(k^2)$ times in T or P is almost periodic with some period Q

[ChKoW’20]

Key Intermediate Result (Analyze)

Every string P satisfies at least one of

- P has $2k$ disjoint, long breaks c❃✿❁❁❃❋a❃✿❁❁❃❋a
- P is almost periodic. aaacaaaaaaaaacaaaaaaa

Structural Results for Approximate String Matching

[ChKoW'20]

Main Result (Edits)

T, P with $t \le {}^{3}/_{2}\, p$; threshold k, P appears $O(k^2)$ times in T or P is almost periodic with some period Q

[ChKoW'20]

Key Intermediate Result (Analyze)

Every string P satisfies at least one of

- P has $2k$ disjoint, long breaks c❃✿❁❁❃❋a❃✿❁❁❃❋a
- P has disjoint repetitive regions that cover ${}^{3}/_{8}\, P$ ❃❋aaaaaa❃❋ccaccc❃❋
- P is almost periodic. aaacaaaaaaaaacaaaaaaa

Structural Results for Approximate String Matching

Main Result (Edits)

[ChKoW'20]

T, P with $t \le {}^{3}\!/_{2}\, p$; threshold k, P appears $O(k^2)$ times in T or P is almost periodic with some period Q

Key Intermediate Result (Analyze)

[ChKoW'20]

Every string P satisfies at least one of

- P has $2k$ disjoint, long breaks — c❉✿❁❁❉❋a❉✿❁❁❉❋a
- P has disjoint repetitive regions that cover ${}^{3}\!/_{8}\, P$ — ❉❋aaaaaa❉❋ccaccc❉❋
- P is almost periodic. — aaacaaaaaaaaacaaaaaaa

Analyze implies Main Result:

Almost periodicity ⇝ as in Main Result

Breaks, repetitive regions ⇝ good filter (as in [Cole, Hariharan'98])

Key Intermediate Result (Analyze)

[ChKoW'20]

Every string P satisfies at least one of

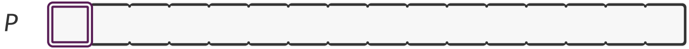
- P has $2k$ disjoint, long breaks c✻✿❁❁✻✻a✻✿❁❁✻✻a
- P has disjoint repetitive regions that cover $^3/_8$ P ✻✻aaaaaa✻✻ccaccc✻✻
- P is almost periodic. aaacaaaaaaaaacaaaaaaa

Key Intermediate Result (Analyze)

[ChKoW'20]

Every string P satisfies at least one of

- P has $2k$ disjoint, long breaks — c❃✿❁❁❃❃a❃✿❁❁❃❃a
- P has disjoint repetitive regions that cover $^3/_8\ P$ — ❃❃aaaaaa❃❃ccaccc❃❃
- P is almost periodic. — aaacaaaaaaaaacaaaaaaa



- Process P from left to right, $p/8k$ new characters at a time.

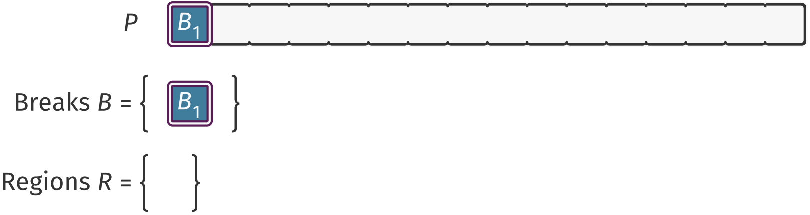
Structural Results for Approximate String Matching

Key Intermediate Result (Analyze)

[ChKoW'20]

Every string P satisfies at least one of

- P has $2k$ disjoint, long breaks c✾✿❁❁✾❋a✾✿❁❁✾❋a
- P has disjoint repetitive regions that cover ${}^{3}/_{8}$ P ✾❋aaaaaa✾❋ccaccc✾❋
- P is almost periodic. aaacaaaaaaaaacaaaaaaa



- If a fragment is a break, add it to the found breaks.

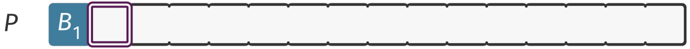
Structural Results for Approximate String Matching

[ChKoW'20]

Key Intermediate Result (Analyze)

Every string P satisfies at least one of

- P has $2k$ disjoint, long breaks   c✽✿❁❁✽✽a✽✿❁❁✽✽a
- P has disjoint repetitive regions that cover $^3/_8$ P ❃❃aaaaaa❃❃ccaccc❃❃
- P is almost periodic.   aaacaaaaaaaaacaaaaaaa



Breaks B = { B_1 }

Repetitive Regions R = { }

- Otherwise, find the shortest prefix (longer than $p/8k$) that is a repetitive region.

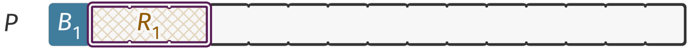
Structural Results for Approximate String Matching

[ChKoW'20]

Key Intermediate Result (Analyze)

Every string P satisfies at least one of

- P has $2k$ disjoint, long breaks c✾✿❁❁✾✾a✾✿❁❁✾✾a
- P has disjoint repetitive regions that cover ${}^{3}/_{8}$ P ✾✾aaaaaa✾✾ccaccc✾✾
- P is almost periodic. aaacaaaaaaaaacaaaaaaa



Breaks $B = \{ B_1 \}$

Repetitive Regions $R = \{ R_1 \}$

- Otherwise, find the shortest prefix (longer than $p/8k$) that is a repetitive region.

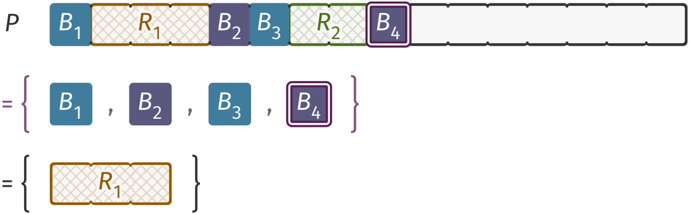
Structural Results for Approximate String Matching

Key Intermediate Result (Analyze)

[ChKoW'20]

Every string P satisfies at least one of

- P has $2k$ disjoint, long breaks — c❃✿❁❁❃❃a❃✿❁❁❃❃a
- P has disjoint repetitive regions that cover $^3/_8$ P — ❃❃aaaaaa❃❃ccaccc❃❃
- P is almost periodic. — aaacaaaaaaaaacaaaaaaa



Breaks $B = \{B_1, B_2, B_3, B_4\}$

Repetitive Regions $R = \{R_1\}$

- If we found $2k$ breaks, return the breaks.

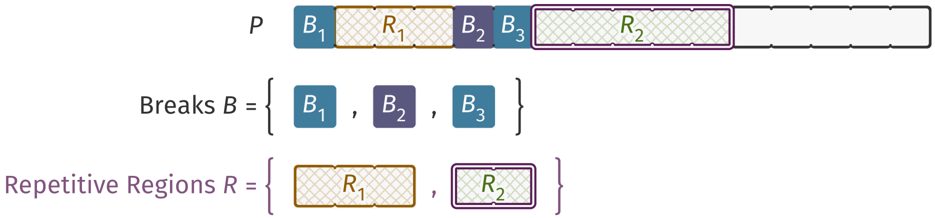
Structural Results for Approximate String Matching

Key Intermediate Result (Analyze)

[ChKoW'20]

Every string P satisfies at least one of

- P has $2k$ disjoint, long breaks — c❃✿❁❁❃❋a❃✿❁❁❃❋a
- P has disjoint repetitive regions that cover $^3/_8$ P — ❃❋aaaaaa❃❋ccaccc❃❋
- P is almost periodic. — aaacaaaaaaaaacaaaaaaa



P
B_1
R_1
B_2
B_3
R_2
Breaks B = { B_1 , B_2 , B_3 }
Repetitive Regions R = { R_1 , R_2 }

- If the total length of the repetitive regions is > $3/8 \cdot p$, return the repetitive regions.

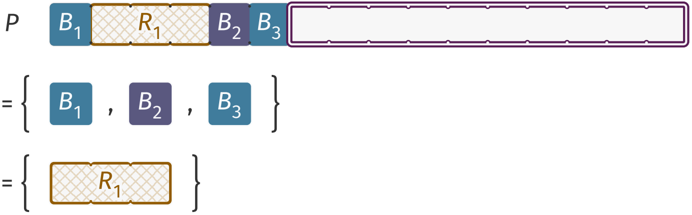
Structural Results for Approximate String Matching

Key Intermediate Result (Analyze)

[ChKoW'20]

Every string P satisfies at least one of

- P has $2k$ disjoint, long breaks — c❃✿❁❁❃❃a❃✿❁❁❃❃a
- P has disjoint repetitive regions that cover $^3/_8$ P — ❃❃aaaaaa❃❃ccaccc❃❃
- P is almost periodic. — aaacaaaaaaaaacaaaaaaa



P B_1 R_1 B_2 B_3

Breaks $B = \{ B_1 , B_2 , B_3 \}$

Repetitive Regions $R = \{ R_1 \}$

- If we reach the end of P, try to find a single repetitive region starting from the end.

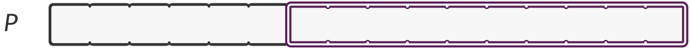
Structural Results for Approximate String Matching

Key Intermediate Result (Analyze)

[ChKoW'20]

Every string P satisfies at least one of

- P has $2k$ disjoint, long breaks — c✾✿❁❁✾✾a✾✿❁❁✾✾a
- P has disjoint repetitive regions that cover $^3/_8\ P$ — ✾✾aaaaaa✾✾ccaccc✾✾
- P is almost periodic. — aaacaaaaaaaaacaaaaaaa



Breaks $B = \{ \quad \}$

Repetitive Regions $R = \{ \quad \}$

- If we reach the end of P, try to find a single repetitive region starting from the end.

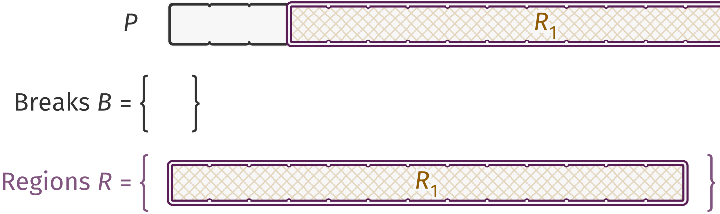
Structural Results for Approximate String Matching

Key Intermediate Result (Analyze)

[ChKoW'20]

Every string P satisfies at least one of

- P has $2k$ disjoint, long breaks — c✽✿❁❁✽✽a✽✿❁❁✽✽a
- P has disjoint repetitive regions that cover ${}^{3}/_{8}$ P — ✽✽aaaaaa✽✽ccaccc✽✽
- P is almost periodic. — aaacaaaaaaaacaaaaaaa



- If we found a repetitive region, return it.

Structural Results for Approximate String Matching

Key Intermediate Result (Analyze)

[ChKoW’20]

Every string P satisfies at least one of

- P has $2k$ disjoint, long breaks c❉✿❁❁❉❉a❉✿❁❁❉❉a
- P has disjoint repetitive regions that cover ${}^{3}/_{8}$ P ❉❉aaaaaa❉❉ccaccc❉❉
- P is almost periodic. aaacaaaaaaaaacaaaaaaa

P

Breaks $B = \{ \quad \}$

Repetitive Regions $R = \{ \quad \}$

- If we again don’t obtain a repetitive region, P is almost periodic.

Structural Results for Approximate String Matching

[ChKoW'20]

Key Intermediate Result (Analyze) ✓

Every string P satisfies at least one of

- P has $2k$ disjoint, long breaks — c❋✿❁❁❋❋a❋✿❁❁❋❋a
- P has disjoint repetitive regions that cover $^3/_8$ P — ❋❋aaaaaa❋❋ccaccc❋❋
- P is almost periodic. — aaacaaaaaaaaacaaaaaaa

New Insights, Same Old Problems

How do we turn our insights
into faster algorithms?

New Insights, Same Old Problems

How do we turn our insights into faster algorithms?

Need to tackle three cases.

- P contains $2k$ disjoint breaks;
- P contains disjoint repetitive regions R_i;
- P is almost periodic

New Insights, Same Old Problems

How do we turn our insights into faster algorithms?

Need to tackle ~~three~~ two cases.

- P contains $2k$ disjoint breaks;
- ~~P contains disjoint repetitive regions R_i;~~ ⇝ Follows from the other two cases.
- P is almost periodic

New Insights, Same Old Problems

How do we turn our insights into faster algorithms?

Need to tackle ~~three~~ one case.

- ~~P contains $2k$ disjoint breaks;~~ ⇝ Adaption of [Cole,Hariharan'98] yields $O(|T| + |T|/|P| \cdot k^3)$.
- ~~P contains disjoint repetitive regions R_i;~~ ⇝ Follows from the other two cases.
- P is almost periodic

Inter-University Research Institute Corporation
Research Organization of Information and Systems
National Institute of Informatics

New Insights, Same Old Problems

How do we turn our insights into faster algorithms?

Need to tackle ~~three~~ one case.

- ~~P contains $2k$ disjoint breaks;~~ ⇝ Adaption of [Cole,Hariharan'98] yields $O(|T| + |T|/|P| \cdot k^3)$.
- ~~P contains disjoint repetitive regions R_i;~~ ⇝ Follows from the other two cases.
- P is almost periodic

[ChKoW'22]

Reduction to Periodic Patterns

Algorithm for almost periodic case in time $\tilde{O}(|T| + k^a \cdot |T|/|P|)$, for $a \geq 3$

$\implies$ Algorithm for general case in time $\tilde{O}(|T| + k^a \cdot |T|/|P|)$

New Insights, Same Old Problems

[ChKoW'22]

Reduction to Periodic Patterns

Algorithm for almost periodic case in time $\tilde{O}(|T| + k^a \cdot |T|/|P|)$, for $a \geq 3$

$\implies$ Algorithm for general case in time $\tilde{O}(|T| + k^a \cdot |T|/|P|)$

Need to tackle: P is almost periodic

New Insights, Same Old Problems

Reduction to Periodic Patterns

[ChKo**W**'22]

Algorithm for almost periodic case in time $\tilde{O}(|T| + k^a \cdot |T|/|P|)$, for $a \geq 3$

$\Longrightarrow$ Algorithm for general case in time $\tilde{O}(|T| + k^a \cdot |T|/|P|)$

Need to tackle: P is almost periodic

- In [ChKo**W**'20], use elaborate marking scheme for $O(|T| + |T|/|P| \cdot k^4)$ algo

  $\leadsto$ Not faster than [Cole,Hariharan'98]

New Insights, Same Old Problems

Reduction to Periodic Patterns

[ChKoW'22]

Algorithm for almost periodic case in time $\tilde{O}(|T| + k^a \cdot |T|/|P|)$, for $a \geq 3$

$\Longrightarrow$ Algorithm for general case in time $\tilde{O}(|T| + k^a \cdot |T|/|P|)$

Need to tackle: P is almost periodic

- In [ChKoW'20], use elaborate marking scheme for $O(|T| + |T|/|P| \cdot k^4)$ algo

  $\rightsquigarrow$ Not faster than [Cole,Hariharan'98]
- In [ChKoW'22], trade-off between
  - Refinement of algorithm from [ChKoW'20]
  - New algorithm based on “Seaweed Technology” of [Tiskin'10,'15]

  $\rightsquigarrow$ Yields $O(|T| + |T|/|P| \cdot k^{3.5})$ algorithm

New Insights, Same Old Problems

[ChKoW’22]

Reduction to Periodic Patterns

Algorithm for almost periodic case in time $\tilde{O}(|T| + k^a \cdot |T|/|P|)$, for $a \geq 3$
$\implies$ Algorithm for general case in time $\tilde{O}(|T| + k^a \cdot |T|/|P|)$

Need to tackle: *P* is almost periodic

- In [ChKo**W**’20], use elaborate marking scheme for $O(|T| + |T|/|P| \cdot k^4)$ algo
  $\rightsquigarrow$ Not faster than [Cole,Hariharan’98]
- In [ChKo**W**’22], trade-off between
  - Refinement of algorithm from [ChKo**W**’20]
  - ~~New algorithm based on “Seaweed Technology” of [Tiskin’10,’15]~~
    New in [ChKo**W**’25]: Simpler plug-in replacement for “Seaweed Technology” based on SMAWK (efficient (min, +)-multiplication of Monge matrices)

$\rightsquigarrow$ Yields $O(|T| + |T|/|P| \cdot k^{3.5})$ algorithm

Beyond Strings: The PILLAR Model

- All of our algorithms rely on a small set of essential operations:
  - LCP(S, T): Compute the length of the longest common prefix of S and T.
  - LCP^R(S, T): Compute the length of the longest common suffix of S and T.
  - IPM(P, T): Compute all exact matches of P in T.
  - Length(S): Compute the length s of S.
  - Access(S, i): Retrieve the character $S[i]$.
  - Extract(S, ℓ, r): Extract the fragment (or substring) $S[\ell\,..\,r)$ from S.

Main Result (PILLAR Algorithm) [ChKoW'22]

For $t \le {}^{3}/_{2}\, p + k$; ASM can be solved using $O(k^{3.5})$ PILLAR operations

Beyond Strings: The PILLAR Model

- All of our algorithms rely on a small set of essential operations:
  - LCP(S, T): Compute the length of the longest common prefix of S and T.
  - LCP^R(S, T): Compute the length of the longest common suffix of S and T.
  - IPM(P, T): Compute all exact matches of P in T.
  - Length(S): Compute the length s of S.
  - Access(S, i): Retrieve the character $S[i]$.
  - ExtRact(S, ℓ, r): Extract the fragment (or substring) $S[\ell\,..\,r)$ from S.

Main Result (PILLAR Algorithm) [ChKoW'22]

For $t \le {}^{3}/_{2}\, p + k$; ASM can be solved using $O(k^{3.5})$ PILLAR operations

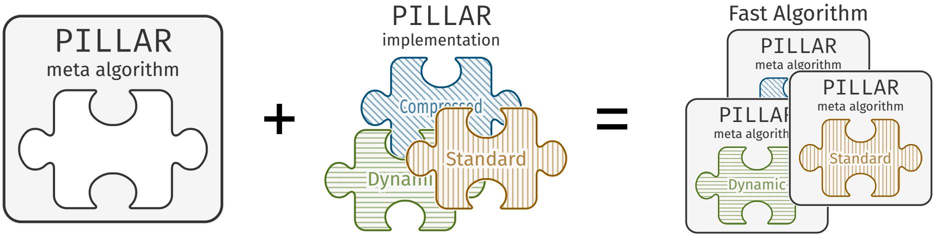
Beyond Strings: The PILLAR Model

- All of our algorithms rely on a small set of essential operations:
  - LCP(S, T): Compute the length of the longest common prefix of S and T.
  - LCP^R(S, T): Compute the length of the longest common suffix of S and T.
  - IPM(P, T): Compute all exact matches of P in T.
  - Length(S): Compute the length s of S.
  - Access(S, i): Retrieve the character $S[i]$.
  - ExtRact(S, ℓ, r): Extract the fragment (or substring) $S[\ell..r)$ from S.

[ChKoW'22]

Main Result (PILLAR Algorithm)

For $t \le {}^{3}/_{2}\, p + k$; ASM can be solved using $O(k^{3.5})$ PILLAR operations



Beyond Strings: The PILLAR Model

- All of our algorithms rely on a small set of essential operations:
  - LCP(S, T): Compute the length of the longest common prefix of S and T.
  - LCP$^R(S, T)$: Compute the length of the longest common suffix of S and T.
  - IPM(P, T): Compute all exact matches of P in T.
  - Length(S): Compute the length s of S.
  - Access(S, i): Retrieve the character $S[i]$.
  - ExtRact(S, ℓ, r): Extract the fragment (or substring) $S[\ell..r)$ from S.

[ChKoW'22]

Main Result (PILLAR Algorithm)

For $t \le {}^3\!/_2\, p + k$; ASM can be solved using $O(k^{3.5})$ PILLAR operations

- In [ChKoW'20]: efficient PILLAR implementations for
  - Standard setting (each op $O(1)$ after global $O(t)$ preprocessing)
  - Fully-compressed setting (T, P grammar-compressed to size n; each op $\tilde{O}(1)$ after $\tilde{O}(n)$ prep.)
    (In this setting, already [ChKoW'20] obtained faster algorithms)
  - Dynamic Setting

Beyond Strings: The PILLAR Model

- All of our algorithms rely on a small set of essential operations:
  - LCP(S, T): Compute the length of the longest common prefix of S and T.
  - LCP$^R(S, T)$: Compute the length of the longest common suffix of S and T.
  - IPM(P, T): Compute all exact matches of P in T.
  - Length(S): Compute the length s of S.
  - Access(S, i): Retrieve the character $S[i]$.
  - ExtRact(S, ℓ, r): Extract the fragment (or substring) $S[\ell\,..\,r)$ from S.

Main Result (PILLAR Algorithm) [ChKoW'22]

For $t \le {}^3\!/_2\, p + k$; ASM can be solved using $O(k^{3.5})$ PILLAR operations

- In [ChKo**W**'20]: efficient PILLAR implementations for
  - Standard setting (each op $O(1)$ after global $O(t)$ preprocessing)
  - Fully-compressed setting (T, P grammar-compressed to size n; each op $\tilde{O}(1)$ after $\tilde{O}(n)$ prep.)
    (In this setting, already [ChKo**W**'20] obtained faster algorithms)
  - Dynamic Setting
- Since then: quantum setting (more details soon!), packed setting, read-only setting, ...
  PILLAR algorithms for other related problems, etc. [ChKo**W**'20] has > 60 citations by now

Beyond Strings: The PILLAR Model

- All of our algorithms rely on a small set of essential operations:
  - LCP(S, T): Compute the length of the longest common prefix of S and T.
  - LCP^R(S, T): Compute the length of the longest common suffix of S and T.
  - IPM(P, T): Compute all exact matches of P in T.
  - Length(S): Compute the length s of S.
  - Access(S, i): Retrieve the character $S[i]$.
  - ExtRact(S, ℓ, r): Extract the fragment (or substring) $S[\ell\,..\,r)$ from S.

Main Result (PILLAR Algorithm) [ChKoW'22]

For $t \le {}^3\!/_2\, p + k$; ASM can be solved using $O(k^{3.5})$ PILLAR operations

- In [ChKoW'20]: efficient PILLAR implementations for
  - Standard setting (each op $O(1)$ after global $O(t)$ preprocessing)
  - Fully-compressed setting (T, P grammar-compressed to size n; each op $\tilde{O}(1)$ after $\tilde{O}(n)$ prep.)
    (In this setting, already [ChKoW'20] obtained faster algorithms)
  - Dynamic Setting
- Since then: quantum setting (more details soon!), packed setting, read-only setting, ...
  PILLAR algorithms for other related problems, etc. [ChKoW'20] has > 60 citations by now
- Bottom-line, useful design principle: meta algorithm (PILLAR) + simple-to-solve subproblems

Spotlight Extension: Quantum Algorithms for ASM

early 1980's

Approximate String Matching

Given: text T, pattern P, integer k; Find: (starting pos. of) substrings of T at edit distance ≤ k to P.

- P/T given as oracle ⇝ queries possible in superposition
- Query complexity $Q(n)$: number of oracle queries
- Time complexity $T(n)$: number of elementary gates

Approximate String Matching

early 1980’s

Given: text T, pattern P, integer k; Find: (starting pos. of) substrings of T at edit distance ≤ k to P.

- P/T given as oracle ⇝ queries possible in superposition
- Query complexity $Q(n)$: number of oracle queries
- Time complexity $T(n)$: number of elementary gates
- Goal: algorithms with $Q(n) \ll n$ and $T(n) \ll n$

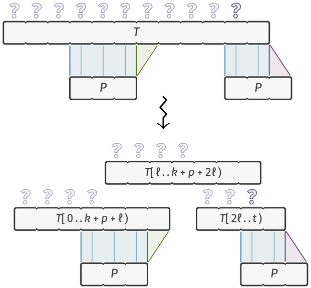
early 1980's

Approximate String Matching

Given: text T, pattern P, integer k; Find: (starting pos. of) substrings of T at edit distance $\leq k$ to P.

- P/T given as oracle ⇝ queries possible in superposition
- Query complexity $Q(n)$: number of oracle queries
- Time complexity $T(n)$: number of elementary gates
- Goal: algorithms with $Q(n) \ll n$ and $T(n) \ll n$

Toy example: “Standard Trick” for ASM decision version (∃? occ)
Split T into overlapping fragments of len $\ell + p + k$
⇝ $O(t/\ell)$ instances, each “responsible” for its first ℓ positions
⇝ Classically: $O(t/\ell)$ time overhead (evaluate each inst. 1 by 1)



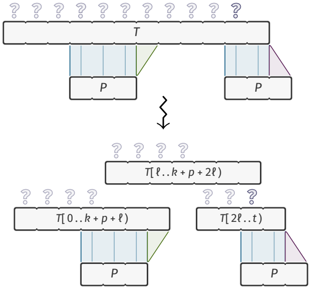
Approximate String Matching

early 1980's

Given: text T, pattern P, integer k; Find: (starting pos. of) substrings of T at edit distance $\leq k$ to P.

- P/T given as oracle ⇝ queries possible in superposition
- Query complexity $Q(n)$: number of oracle queries
- Time complexity $T(n)$: number of elementary gates
- Goal: algorithms with $Q(n) \ll n$ and $T(n) \ll n$

Toy example: "Standard Trick" for ASM decision version (∃? occ)
Split T into overlapping fragments of len $\ell + p + k$
⇝ $O(t/\ell)$ instances, each "responsible" for its first ℓ positions
⇝ Classically: $O(t/\ell)$ time overhead (evaluate each inst. 1 by 1)
Quantum setting: Use amplitude amp. / Grover's algo.
$\tilde{O}(\sqrt{t/\ell})$ time and $O(\sqrt{t/\ell})$ evaluations of single inst.



Amplitude Amplification

[Gro96, BHMT02]

Given function $f : [0..n) \to \{0,1\}$
Can (wp $\geq 2/3$) obtain $x \in f^{-1}(1)$ in $\tilde{O}(\sqrt{n})$ time + $O(\sqrt{n})$ queries to f

Plugging Together a First Algorithm

early 1980's

Approximate String Matching

Given: text T, pattern P, integer k; Find: (starting pos. of) substrings of T at edit distance $\leq k$ to P.

- P/T given as oracle $\rightsquigarrow$ queries possible in superposition
- Goal: algorithms with query complexity $Q(n) \ll n$ and time complexity $T(n) \ll n$

Idea: Use existing quantum algorithms to implement PILLAR operations.

Plugging Together a First Algorithm

Approximate String Matching

early 1980's

Given: text T, pattern P, integer k; Find: (starting pos. of) substrings of T at edit distance $\leq k$ to P.

- P/T given as oracle ⇝ queries possible in superposition
- Goal: algorithms with query complexity $Q(n) \ll n$ and time complexity $T(n) \ll n$

Idea: Use existing quantum algorithms to implement PILLAR operations.

LC**P**(S,T); **I**PM(P,T); **L**ength(S); **A**ccess(S,i); Ext**R**act(S,ℓ,r)

PILLAR **Algorithm**

[ChKoW'22]

For $t \leq {}^{3}/_{2}\, p + k$;
ASM can be solved
using $O(k^{3.5})$ PILLAR operations

Plugging Together a First Algorithm

early 1980's

Approximate String Matching

Given: text T, pattern P, integer k; Find: (starting pos. of) substrings of T at edit distance $\leq k$ to P.

- P/T given as oracle ⇝ queries possible in superposition
- Goal: algorithms with query complexity $Q(n) \ll n$ and time complexity $T(n) \ll n$

Idea: Use existing quantum algorithms to implement PILLAR operations.

LC**P**(S,T); **I**PM(P,T); **L**ength(S); **A**ccess(S,i); Ext**R**act(S,ℓ,r)

PILLAR **Algorithm**

[ChKoW'22]

For $t \leq {}^{3}\!/_{2}\, p + k$;
ASM can be solved
using $O(k^{3.5})$ PILLAR operations

Quantum PILLAR **Impl**

implied by [HV'03]

For length-n strings,
can implement all PILLAR ops
in $\tilde{O}(\sqrt{n})$ time and queries

Yields $\tilde{O}(t/p \cdot k^{3.5} \cdot \sqrt{p})$ quantum time algo for ASM; ($\tilde{O}(\sqrt{t/p} \cdot k^{3.5} \cdot \sqrt{p})$ for $\exists$?) (algos are correct whp.)

Plugging Together a First Algorithm

early 1980's

Approximate String Matching

Given: text T, pattern P, integer k; Find: (starting pos. of) substrings of T at edit distance $\leq k$ to P.

- P/T given as oracle ⇝ queries possible in superposition
- Goal: algorithms with query complexity $Q(n) \ll n$ and time complexity $T(n) \ll n$

Idea: Use existing quantum algorithms to implement PILLAR operations.

LC**P**(S,T); **I**PM(P,T); **L**ength(S); **A**ccess(S,i); Ext**R**act(S,ℓ,r)

PILLAR Algorithm [ChKoW'22]

For $t \leq {}^{3}\!/_{2}\, p + k$;
ASM can be solved
using $O(k^{3.5})$ PILLAR operations

Quantum PILLAR Impl implied by [HV'03]

For length-n strings,
can implement all PILLAR ops
in $\tilde{O}(\sqrt{n})$ time and queries

Yields $\tilde{O}(t/p \cdot k^{3.5} \cdot \sqrt{p})$ quantum time algo for ASM; ($\tilde{O}(\sqrt{t/p} \cdot k^{3.5} \cdot \sqrt{p})$ for $\exists$?) (algos are correct whp.)

Are these algorithms optimal? ⇝ No.

If You Want to Be Fast, Take a Detour

Approximate String Matching

early 1980's

Given: text T, pattern P, integer k; Find: (starting pos. of) substrings of T at edit distance ≤ k to P.

- P/T given as oracle ⇝ queries possible in superposition
- Goal: algorithms with query complexity $Q(n) \ll n$ and time complexity $T(n) \ll n$

Challenge: query as few char's as possible, but enough to recover all k-edit occ's of P in T

If You Want to Be Fast, Take a Detour

early 1980's

Approximate String Matching

Given: text T, pattern P, integer k; Find: (starting pos. of) substrings of T at edit distance $\leq k$ to P.

- P/T given as oracle ⇝ queries possible in superposition
- Goal: algorithms with query complexity $Q(n) \ll n$ and time complexity $T(n) \ll n$

Challenge: query as few char's as possible, but enough to recover all k-edit occ's of P in T

⇝ k-edit-aware lossy compression of P and T (think: filter + discarding filtered-out parts)

Approximate String Matching

early 1980's

Given: text T, pattern P, integer k; Find: (starting pos. of) substrings of T at edit distance $\leq k$ to P.

- P/T given as oracle ⇝ queries possible in superposition
- Goal: algorithms with query complexity $Q(n) \ll n$ and time complexity $T(n) \ll n$

Challenge: query as few char's as possible, but enough to recover all k-edit occ's of P in T

⇝ k-edit-aware lossy compression of P and T (think: filter + discarding filtered-out parts)

In [KoN**W**'24, KoN**W**'25] for T and P with $t \leq {}^{3}/_{2}\, p + k$:

- Show: any k-edit-aware lossy compr. of P and T needs $\Omega(k \log(t/k))$ bits

If You Want to Be Fast, Take a Detour

Approximate String Matching

early 1980's

Given: text T, pattern P, integer k; Find: (starting pos. of) substrings of T at edit distance $\leq k$ to P.

- P/T given as oracle ⇝ queries possible in superposition
- Goal: algorithms with query complexity $Q(n) \ll n$ and time complexity $T(n) \ll n$

Challenge: query as few char's as possible, but enough to recover all k-edit occ's of P in T

⇝ k-edit-aware lossy compression of P and T (think: filter + discarding filtered-out parts)

In [KoN**W**'24, KoN**W**'25] for T and P with $t \leq {}^3/_2\, p + k$:

- Show: any k-edit-aware lossy compr. of P and T needs $\Omega(k \log(t/k))$ bits
- Obtain k-edit-aware lossy compr. of P and T as $\tilde{O}(k)$ substring equations; computable via quantum algorithm using $\hat{O}(\sqrt{kp})$ queries/time

If You Want to Be Fast, Take a Detour

early 1980's

Approximate String Matching

Given: text T, pattern P, integer k; Find: (starting pos. of) substrings of T at edit distance $\leq k$ to P.

- P/T given as oracle ⇝ queries possible in superposition
- Goal: algorithms with query complexity $Q(n) \ll n$ and time complexity $T(n) \ll n$

Challenge: query as few char's as possible, but enough to recover all k-edit occ's of P in T

⇝ k-edit-aware lossy compression of P and T (think: filter + discarding filtered-out parts)

In [KoN**W**'24, KoN**W**'25] for T and P with $t \leq {}^3/_2\, p + k$:

- Show: any k-edit-aware lossy compr. of P and T needs $\Omega(k \log(t/k))$ bits
- Obtain k-edit-aware lossy compr. of P and T as $\tilde{O}(k)$ substring equations; computable via quantum algorithm using $\hat{O}(\sqrt{kp})$ queries/time
- Give classical $\tilde{O}(b^2)$-time algo to compute sol. to b substr. eqn's as $O(b)$-size grammar

If You Want to Be Fast, Take a Detour

early 1980's

Approximate String Matching

Given: text T, pattern P, integer k; Find: (starting pos. of) substrings of T at edit distance $\leq k$ to P.

- P/T given as oracle ⇝ queries possible in superposition
- Goal: algorithms with query complexity $Q(n) \ll n$ and time complexity $T(n) \ll n$

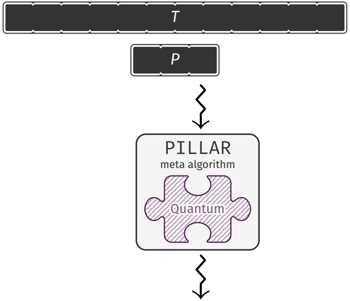
Challenge: query as few char's as possible, but enough to recover all k-edit occ's of P in T
⇝ k-edit-aware lossy compression of P and T (think: filter + discarding filtered-out parts)

In [KoN**W**'24, KoN**W**'25] for T and P with $t \leq {}^{3}/_{2}\, p + k$:

- Show: any k-edit-aware lossy compr. of P and T needs $\Omega(k \log(t/k))$ bits
- Obtain k-edit-aware lossy compr. of P and T as $\tilde{O}(k)$ substring equations;
  computable via quantum algorithm using $\hat{O}(\sqrt{kp})$ queries/time
- Give classical $\tilde{O}(b^2)$-time algo to compute sol. to b substr. eqn's as $O(b)$-size grammar
  ⇝ Can use PILLAR algorithm for fully-compressed setting on lossy compression!

Quantum ASM: Bottom-Lines

- P/T given as oracle ⇝ queries possible in superposition
- Goal: algorithms with query complexity $Q(n) \ll n$ and time complexity $T(n) \ll n$



T

P

PILLAR

meta algorithm

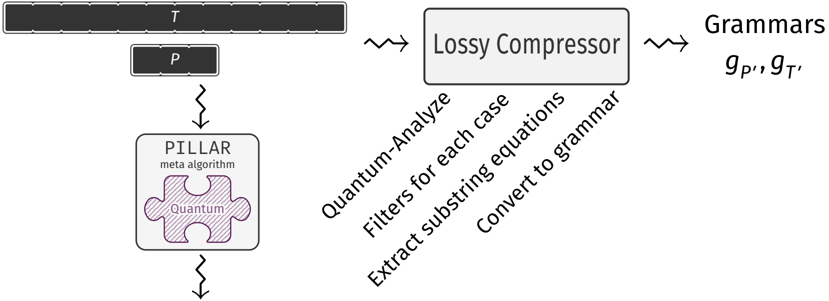
Quantum

Easy Result

ASM on T, P w/ $t \le {}^{3}/_{2}\, p + k$ admits qu-algo using $\tilde{O}(k^{3.5}\sqrt{n})$ queries/time

Quantum ASM: Bottom-Lines

- P/T given as oracle $\rightsquigarrow$ queries possible in superposition
- Goal: algorithms with query complexity $Q(n) \ll n$ and time complexity $T(n) \ll n$



T

P

PILLAR
meta algorithm

Quantum

Lossy Compressor

Grammars
$g_{P'}, g_{T'}$

Quantum-Analyze

Filters for each case

Extract substring equations

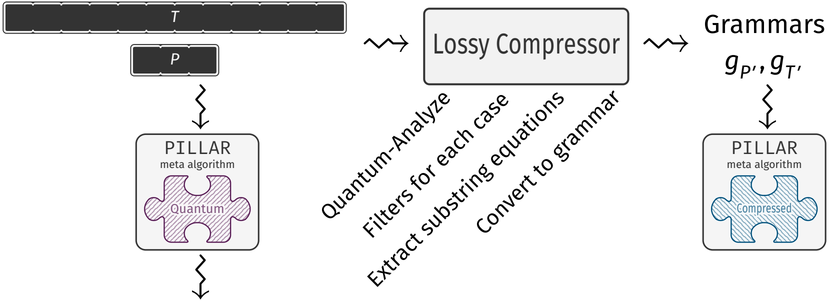
Convert to grammar

Easy Result

ASM on T, P w/ $t \le {}^3/_2\, p + k$ admits qu-algo using $\tilde{O}(k^{3.5}\sqrt{n})$ queries/time

Quantum ASM: Bottom-Lines

- P/T given as oracle ⇝ queries possible in superposition
- Goal: algorithms with query complexity $Q(n) \ll n$ and time complexity $T(n) \ll n$



T

P

PILLAR
meta algorithm
Quantum

Lossy Compressor

Quantum-Analyze

Filters for each case

Extract substring equations

Convert to grammar

Grammars
$g_{P'}, g_{T'}$

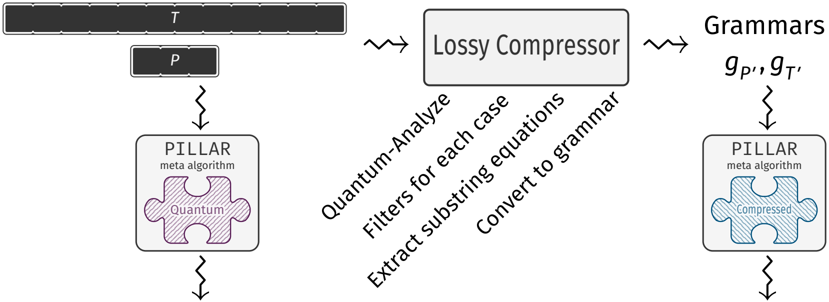
PILLAR
meta algorithm
Compressed

Easy Result

ASM on T, P w/ $t \le {}^{3}/_{2}\, p + k$ admits qu-algo
using $\tilde{O}(k^{3.5}\sqrt{n})$ queries/time

Quantum ASM: Bottom-Lines

- P/T given as oracle ⇝ queries possible in superposition
- Goal: algorithms with query complexity $Q(n) \ll n$ and time complexity $T(n) \ll n$



T

P

Lossy Compressor

Grammars

$g_{P'}, g_{T'}$

PILLAR

meta algorithm

Quantum

Quantum-Analyze

Filters for each case

Extract substring equations

Convert to grammar

PILLAR

meta algorithm

Compressed

Easy Result

ASM on T, P w/ $t \leq {}^{3}\!/_{2}\, p + k$ admits qu-algo using $\tilde{O}(k^{3.5}\sqrt{n})$ queries/time

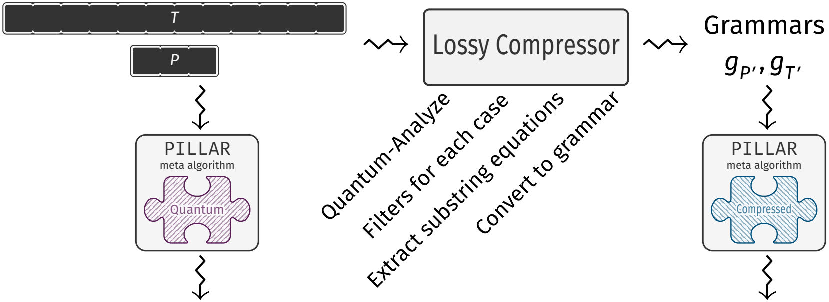
Better Result

[KoNW'24,'25]

ASM on T, P w/ $t \leq {}^{3}\!/_{2}\, p + k$ admits qu-algo using $\hat{O}(\sqrt{kn})$ queries + $\hat{O}(\sqrt{kn} + k^{3.5})$ time

Quantum ASM: Bottom-Lines

- P/T given as oracle ⇝ queries possible in superposition
- Goal: algorithms with query complexity $Q(n) \ll n$ and time complexity $T(n) \ll n$



T

P

PILLAR
meta algorithm
Quantum

Lossy Compressor

Quantum-Analyze

Filters for each case

Extract substring equations

Convert to grammar

Grammars

$g_{P'}, g_{T'}$

PILLAR
meta algorithm
Compressed

Easy Result

ASM on T, P w/ $t \le {}^{3}\!/_{2}\, p + k$ admits qu-algo using $\tilde{O}(k^{3.5}\sqrt{n})$ queries/time

Better Result

[KoNW'24,'25]

ASM on T, P w/ $t \le {}^{3}\!/_{2}\, p + k$ admits qu-algo using $\hat{O}(\sqrt{kn})$ queries + $\hat{O}(\sqrt{kn} + k^{3.5})$ time

⇝ Queries are (near) optimal [KoNW'24], time overhead from PILLAR (but future-proof)

Using Analyze + PILLAR for Obtaining New Algorithms 2

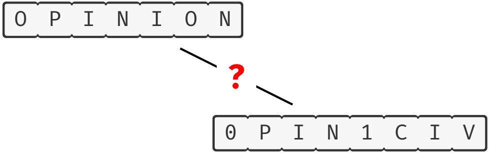
Spotlight Extension: String Matching with Weighted Edits

An Example

How similar are two strings X and Y?

An Example

How similar are two strings *X* and *Y*?



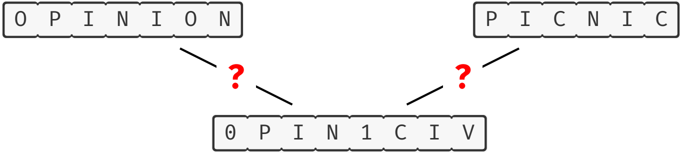
O P I N I O N

?

0 P I N 1 C I V

An Example

How similar are two strings *X* and *Y*?



O P I N I O N

P I C N I C

?

?

0 P I N 1 C I V

Edit Distance, Once More

Edit Distance $\mathrm{ED}(X,Y)$

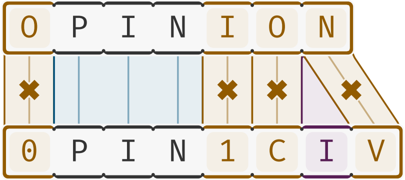
Min number of character insertions, deletions, and substitutions that transform X to Y

Edit Distance, Once More

Edit Distance

ED(X, Y)

Min number of character insertions, deletions, and substitutions that transform X to Y



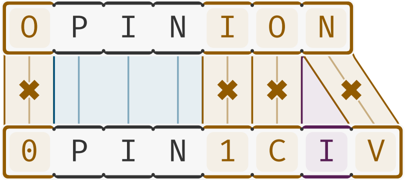
ED(0PIN1CIV, OPINION) = 5

Edit Distance, Once More

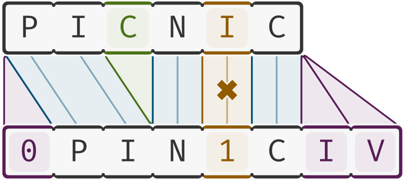
ED(X, Y)

Edit Distance

Min number of character insertions, deletions, and substitutions that transform X to Y



ED(0PIN1CIV, OPINION) = 5



ED(0PIN1CIV, PICNIC) = 5

Edit Distance, Once More

$\mathrm{ED}^w(X, Y)$

Weighted Edit Distance

Min cost of transforming X to Y using character edits, where:

- inserting y costs $w(\varepsilon, y)$;
- deleting x costs $w(x, \varepsilon)$;
- substituting x for y costs $w(x, y)$.

$w(\mathtt{0}, \mathtt{O}) := 1 \quad w(\mathtt{1}, \mathtt{I}) := 1 \quad w(\mathtt{C}, \mathtt{O}) := 1 \qquad w(*, *) := 2 \quad w(*, \varepsilon) := 1 \quad w(\varepsilon, *) := 10$

Edit Distance, Once More

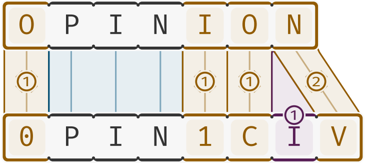
$\mathrm{ED}^w(X, Y)$

Weighted Edit Distance

Min cost of transforming X to Y using character edits, where:

- inserting y costs $w(\varepsilon, y)$;
- deleting x costs $w(x, \varepsilon)$;
- substituting x for y costs $w(x, y)$.

$w(\mathtt{0}, \mathtt{O}) := 1$ $w(\mathtt{1}, \mathtt{I}) := 1$ $w(\mathtt{C}, \mathtt{O}) := 1$ $w(*, *) := 2$ $w(*, \varepsilon) := 1$ $w(\varepsilon, *) := 10$



$\mathrm{ED}^w(\mathtt{0PIN1CIV}, \mathtt{OPINION}) = 6$

Edit Distance, Once More

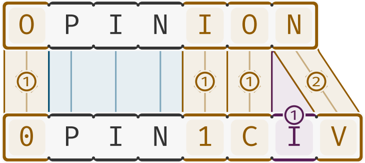
$\mathrm{ED}^w(X, Y)$

Weighted Edit Distance

Min cost of transforming X to Y using character edits, where:

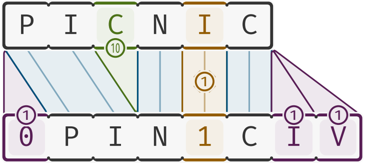
- inserting y costs $w(\varepsilon, y)$;
- deleting x costs $w(x, \varepsilon)$;
- substituting x for y costs $w(x, y)$.

$w(\mathtt{0}, \mathrm{O}) := 1 \quad w(\mathtt{1}, \mathrm{I}) := 1 \quad w(\mathrm{C}, \mathrm{O}) := 1$



$\mathrm{ED}^w(\mathtt{0PIN1CIV}, \mathtt{OPINION}) = 6$

$w(*, *) := 2 \quad w(*, \varepsilon) := 1 \quad w(\varepsilon, *) := 10$



$\mathrm{ED}^w(\mathtt{0PIN1CIV}, \mathtt{PICNIC}) \le 14$

Edit Distance, Once More

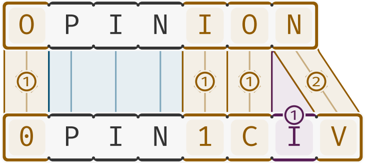
$\mathrm{ED}^w(X, Y)$

Weighted Edit Distance

Min cost of transforming X to Y using character edits, where:

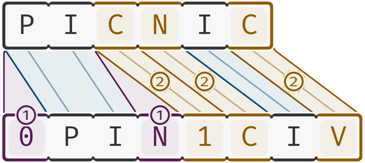
- inserting y costs $w(\varepsilon, y)$;
- deleting x costs $w(x, \varepsilon)$;
- substituting x for y costs $w(x, y)$.

$w(\mathtt{0}, \mathtt{O}) := 1$ $w(\mathtt{1}, \mathtt{I}) := 1$ $w(\mathtt{C}, \mathtt{O}) := 1$



$\mathrm{ED}^w(\mathtt{0PIN1CIV}, \mathtt{OPINION}) = 6$

$w(*, *) := 2$ $w(*, \varepsilon) := 1$ $w(\varepsilon, *) := 10$



$\mathrm{ED}^w(\mathtt{0PIN1CIV}, \mathtt{PICNIC}) = 8$

Edit Distance, Once More

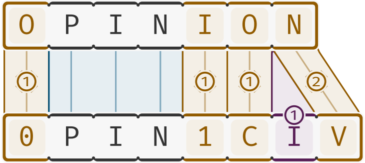
$\mathrm{ED}^w(X, Y)$

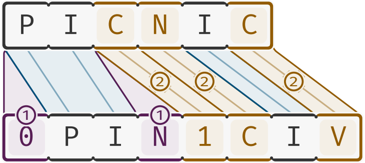
Weighted Edit Distance

Min cost of transforming X to Y using character edits, where:

- inserting y costs $w(\varepsilon, y)$;
- deleting x costs $w(x, \varepsilon)$;
- substituting x for y costs $w(x, y)$.

$w(\mathtt{0}, \mathtt{O}) := 1 \quad w(\mathtt{1}, \mathtt{I}) := 1 \quad w(\mathtt{C}, \mathtt{O}) := 1 \qquad w(*, *) := 2 \quad w(*, \varepsilon) := 1 \quad w(\varepsilon, *) := 10$





$\mathrm{ED}^w(\mathtt{0PIN1CIV}, \mathtt{OPINION}) = 6$

$\mathrm{ED}^w(\mathtt{0PIN1CIV}, \mathtt{PICNIC}) = 8$

Justified Assumption: w is normalized, $w(x, y) \geq 1$ for all $x \neq y$.

Edit Distance, Once More

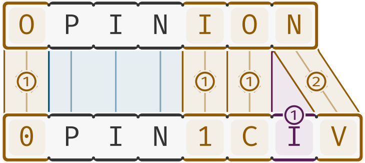
$\mathrm{ED}^{w}(X, Y)$

Weighted Edit Distance

Min cost of transforming X to Y using character edits, where:

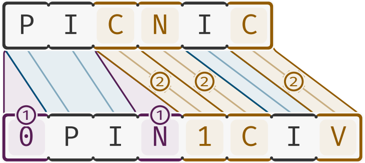
- inserting y costs $w(\varepsilon, y)$;
- deleting x costs $w(x, \varepsilon)$;
- substituting x for y costs $w(x, y)$.

$w(\mathtt{0}, \mathtt{O}) := 1$ $w(\mathtt{1}, \mathtt{I}) := 1$ $w(\mathtt{C}, \mathtt{O}) := 1$



$\mathrm{ED}^{w}(\mathtt{0PIN1CIV}, \mathtt{OPINION}) = 6$

$w(*, *) := 2$ $w(*, \varepsilon) := 1$ $w(\varepsilon, *) := 10$



$\mathrm{ED}^{w}(\mathtt{0PIN1CIV}, \mathtt{PICNIC}) = 8$

Justified Assumption: w is normalized, $w(x, y) \geq 1$ for all $x \neq y$.

Otherwise: could scale weights and condition $\mathrm{ED}^{w}_{\leq k}(X, Y) \leq k$ (e.g. in ASM) becomes meaningless.

Computing String Matching with (Weighted) Edits

Recall crucial subroutine for ASM:

Verify that occurrence starts at given (interval of) position of T

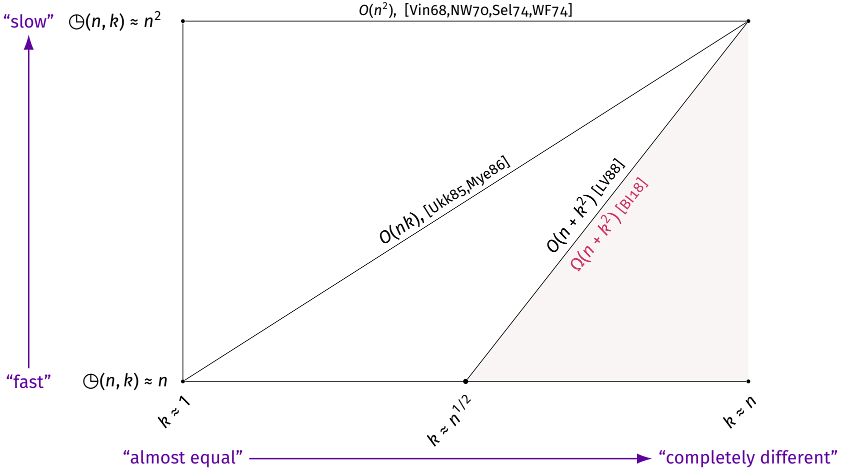
Computing String Matching with (Weighted) Edits

Recall crucial subroutine for ASM:

Verify that occurrence starts at given (interval of) position of T

$\rightsquigarrow$ Need to compute Bounded Weighted Edit Distance

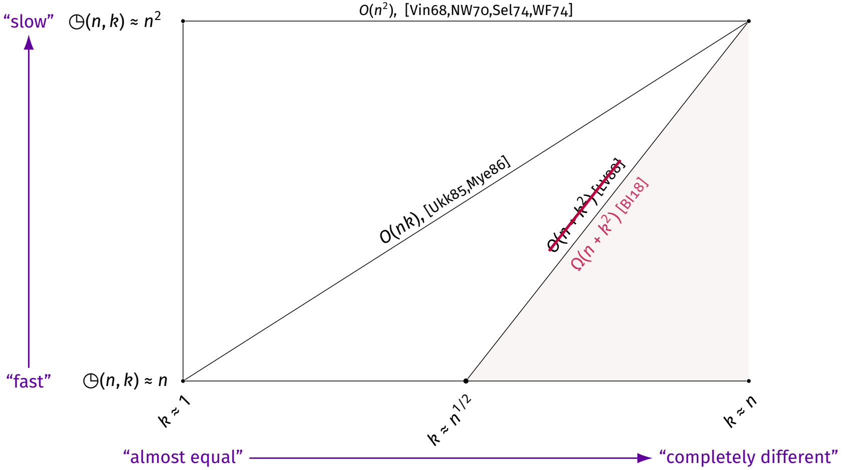
Recall: Algorithms for (Bounded) Edit Distance / Verify



Existing algorithms for Edit Distance $\mathrm{ED}(X, Y)$, where $|X|, |Y| \leq n$

What about Weighted Edit Distance?

Algorithms for (Bounded) Weighted Edit Distance / Weighted Verify



$O(n^2)$, [Vin68,NW70,Sel74,WF74]

"slow" ◷$(n, k) \approx n^2$

"fast" ◷$(n, k) \approx n$

$O(nk)$, [Ukk85,Mye86]

~~$O(n + k^2)$ [LV88]~~

$\Omega(n + k^2)$ [BI18]

$k \approx 1$

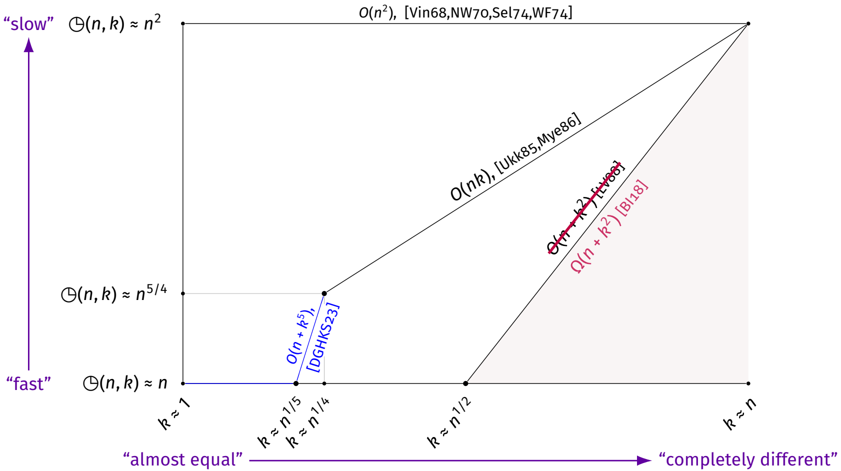
$k \approx n^{1/2}$

$k \approx n$

"almost equal" → "completely different"

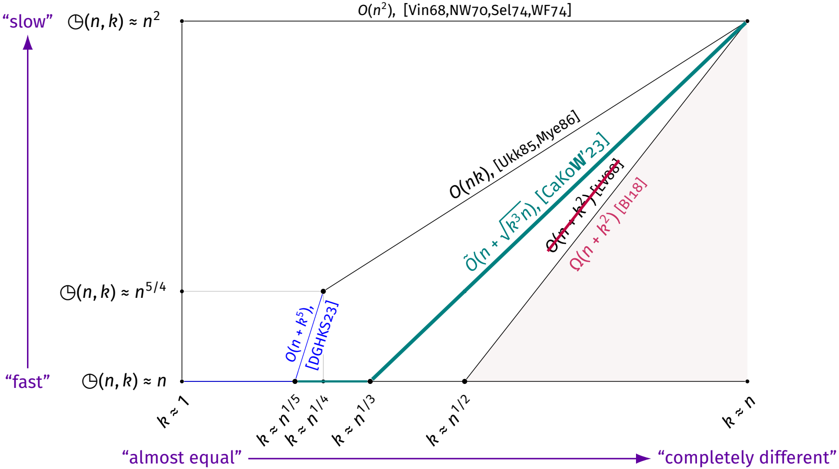
Existing algorithms for Weighted Edit Distance $\mathrm{ED}^w(X, Y)$, where $|X|, |Y| \leq n$

Algorithms for (Bounded) Weighted Edit Distance / Weighted Verify



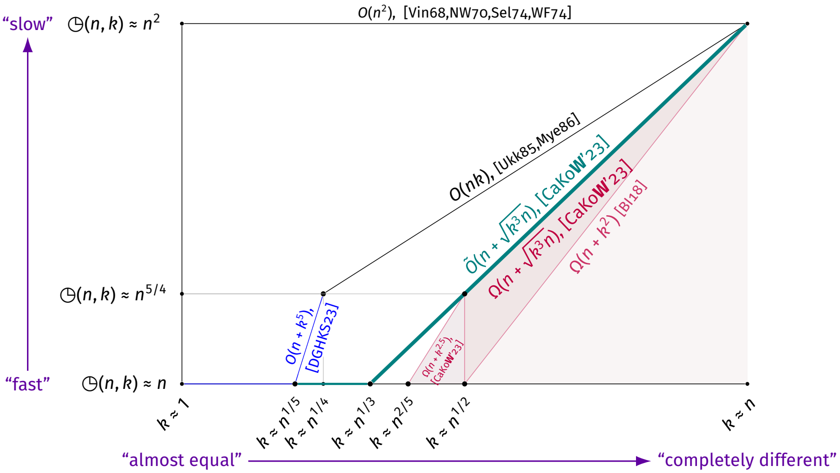
Existing algorithms for Weighted Edit Distance $\mathrm{ED}^w(X,Y)$, where $|X|,|Y| \leq n$

Algorithms for (Bounded) Weighted Edit Distance / Weighted Verify



Existing algorithms for Weighted Edit Distance $\mathrm{ED}^w(X, Y)$, where $|X|, |Y| \le n$

Algorithms for (Bounded) Weighted Edit Distance / Weighted Verify



Existing algorithms for Weighted Edit Distance $\mathrm{ED}^w(X, Y)$, where $|X|, |Y| \le n$

Algorithms for (Bounded) Weighted Edit Distance / Weighted Verify

[CaKoW'23]

Main Theorem (Upper Bound)

Strings X, Y each of length at most n

Oracle access to (normalized) weight function w

Can compute $k = \mathrm{ED}^w(X, Y)$ in time $O(n + \sqrt{nk^3} \log^3 n)$

Algorithms for (Bounded) Weighted Edit Distance / Weighted Verify

Main Theorem (Upper Bound)

[CaKoW'23]

Strings X, Y each of length at most n

Oracle access to (normalized) weight function w

Can compute $k = \mathrm{ED}^w(X, Y)$ in time $O(n + \sqrt{nk^3} \log^3 n)$

Main Theorem (Lower Bound)

[CaKoW'23]

Assuming the APSP Hypothesis and for $\sqrt{n} \le k \le n$,

Main Theorem (Upper Bound) is tight (up to $n^{o(1)}$-factors)

Algorithms for (Bounded) Weighted Edit Distance / Weighted Verify

[CaKoW'23]

Main Theorem (Upper Bound)

Strings X, Y each of length at most n

Oracle access to (normalized) weight function w

Can compute $k = \mathrm{ED}^w(X, Y)$ in time $O(n + \sqrt{nk^3} \log^3 n)$

[CaKoW'23]

Main Theorem (Lower Bound)

Assuming the APSP Hypothesis and for $\sqrt{n} \le k \le n$,

Main Theorem (Upper Bound) is tight (up to $n^{o(1)}$-factors)

Can extend UB to verify ≤ k consecutive positions

Algorithms for (Bounded) Weighted Edit Distance / Weighted Verify

[CaKo**W**’23]

Main Theorem (Upper Bound)

Strings X, Y each of length at most n

Oracle access to (normalized) weight function w

Can compute $k = \mathrm{ED}^w(X, Y)$ in time $O(n + \sqrt{nk^3}\log^3 n)$

[CaKo**W**’23]

Main Theorem (Lower Bound)

Assuming the APSP Hypothesis and for $\sqrt{n} \le k \le n$,

Main Theorem (Upper Bound) is tight (up to $n^{o(1)}$-factors)

Can extend UB to verify $\le k$ consecutive positions

$\rightsquigarrow$ Good enough for $\tilde{O}(t + t/p \cdot k^4)$ algo for weighted ASM (long, boring) [ChKo**W**’25]

Algorithms for (Bounded) Weighted Edit Distance / Weighted Verify

[CaKoW'23]

Main Theorem (Upper Bound)

Strings X, Y each of length at most n

Oracle access to (normalized) weight function w

Can compute $k = \mathrm{ED}^w(X, Y)$ in time $O(n + \sqrt{nk^3} \log^3 n)$

[CaKoW'23]

Main Theorem (Lower Bound)

Assuming the APSP Hypothesis and for $\sqrt{n} \le k \le n$,

Main Theorem (Upper Bound) is tight (up to $n^{o(1)}$-factors)

Can extend UB to verify $\le k$ consecutive positions

$\rightsquigarrow$ Good enough for $\tilde{O}(t + t/p \cdot k^4)$ algo for weighted ASM (long, boring) [ChKoW'25]

$\rightsquigarrow$ What about $\tilde{O}(kt)$-time for weighted ASM?

With Standard Trick, UB yields $\tilde{O}(t/k \cdot \sqrt{pk^3}) = \tilde{O}(t\sqrt{pk})$

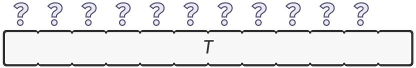
LB rules out $\tilde{O}(kt)$ via better WED algo $\rightsquigarrow$ need different approach

Searching for $\tilde{O}(kt)$-time Algorithms for Weighted ASM

Weighted Approximate String Matching

Given: T, P, k, oracle to w; Find: (starting pos. of) substr. of T at weighted ED ≤ k to P.

Focus: Obtain starting positions of occurrences



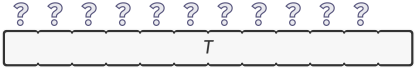
Searching for $\tilde{O}(kt)$-time Algorithms for Weighted ASM

Weighted Approximate String Matching

Given: T, P, k, oracle to w; Find: (starting pos. of) substr. of T at weighted ED ≤ k to P.

Focus: Obtain starting positions of occurrences

Observation: $\mathrm{ED}^w(X, Y) \geq \mathrm{ED}(X, Y)$ (by norm.)



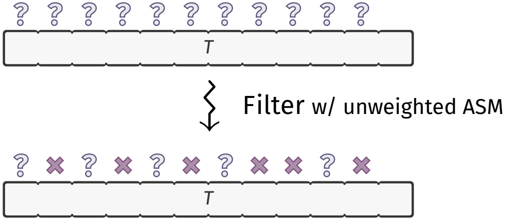
Searching for $\tilde{O}(kt)$-time Algorithms for Weighted ASM

Weighted Approximate String Matching

Given: T, P, k, oracle to w; Find: (starting pos. of) substr. of T at weighted ED ≤ k to P.

Focus: Obtain starting positions of occurrences

Observation: $\mathrm{ED}^w(X,Y) \geq \mathrm{ED}(X,Y)$ (by norm.)
⇝ Use unweighted $O(kn)$-time ASM as Filter



Searching for $\tilde{O}(kt)$-time Algorithms for Weighted ASM

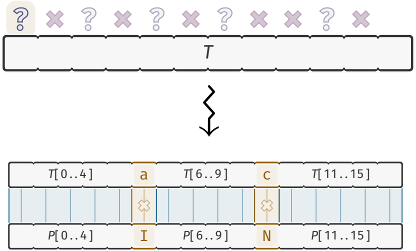
Weighted Approximate String Matching

Given: T, P, k, oracle to w; Find: (starting pos. of) substr. of T at weighted ED ≤ k to P.

Focus: Obtain starting positions of occurrences

Observation: $\mathrm{ED}^w(X, Y) \geq \mathrm{ED}(X, Y)$ (by norm.)
$\rightsquigarrow$ Use unweighted $O(kn)$-time ASM as Filter

If $\mathrm{ED}(P, T[\,a\,..\,b\,]) \leq k$, also get cheap alignment
$P \rightsquigarrow\!\!\!\!\twoheadrightarrow T[\,a\,..\,b\,]$ that matches exactly most of P



Searching for $\tilde{O}(kt)$-time Algorithms for Weighted ASM

Weighted Approximate String Matching

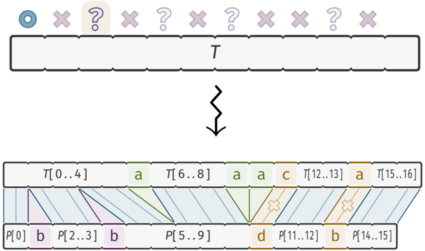
Given: T, P, k, oracle to w; Find: (starting pos. of) substr. of T at weighted ED ≤ k to P.

Focus: Obtain starting positions of occurrences

Observation: $\mathrm{ED}^w(X, Y) \geq \mathrm{ED}(X, Y)$ (by norm.)
$\rightsquigarrow$ Use unweighted $O(kn)$-time ASM as Filter

If $\mathrm{ED}(P, T[\,a\,..\,b\,]) \leq k$, also get cheap alignment
$P \rightsquigarrow\!\!\!\!\!\!\rightarrow T[\,a\,..\,b\,]$ that matches exactly most of P

All such alignments very similar to each other
But how to exploit this fact?



Searching for $\tilde{O}(kt)$-time Algorithms for Weighted ASM

Weighted Approximate String Matching

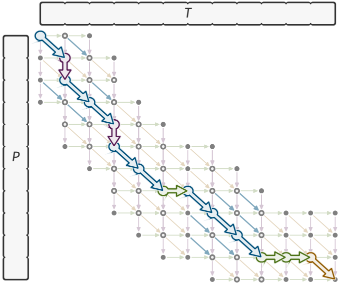
Given: T, P, k, oracle to w; Find: (starting pos. of) substr. of T at weighted ED ≤ k to P.

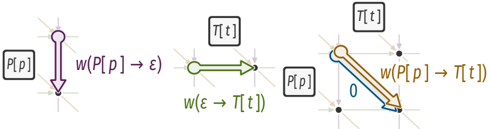
Focus: Obtain starting positions of occurrences

Observation: $\mathrm{ED}^{w}(X, Y) \geq \mathrm{ED}(X, Y)$ (by norm.)
⇝ Use unweighted $O(kn)$-time ASM as Filter

If $\mathrm{ED}(P, T[\,a\,..\,b\,]) \leq k$, also get cheap alignment
$P \rightsquigarrow\!\!\!\rightarrow T[\,a\,..\,b\,]$ that matches exactly most of P

All such alignments very similar to each other
But how to exploit this fact?
⇝ Cast as shortest path prob in alignment graphs





Searching for $\tilde{O}(kt)$-time Algorithms for Weighted ASM

Weighted Approximate String Matching

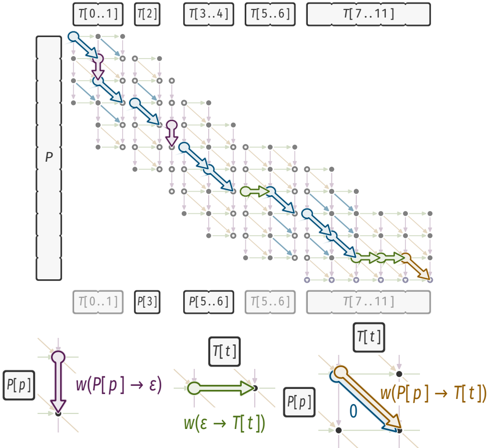
Given: T, P, k, oracle to w; Find: (starting pos. of) substr. of T at weighted ED ≤ k to P.

Focus: Obtain starting positions of occurrences

Observation: $\mathrm{ED}^w(X, Y) \geq \mathrm{ED}(X, Y)$ (by norm.)
⇝ Use unweighted $O(kn)$-time ASM as Filter

If $\mathrm{ED}(P, T[a\,..\,b]) \leq k$, also get cheap alignment
$P \rightsquigarrow\!\!\!\!\!\twoheadrightarrow T[a\,..\,b]$ that matches exactly most of P

All such alignments very similar to each other
But how to exploit this fact?
⇝ Cast as shortest path prob in alignment graphs
⇝ Copy matched parts from (trivial) align. $P \rightsquigarrow\!\!\!\!\!\twoheadrightarrow P$



Searching for $\tilde{O}(kt)$-time Algorithms for Weighted ASM

Weighted Approximate String Matching

Given: T, P, k, oracle to w; Find: (starting pos. of) substr. of T at weighted ED ≤ k to P.

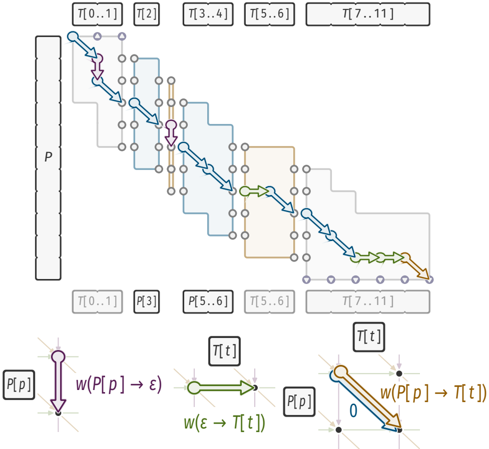
Focus: Obtain starting positions of occurrences

Observation: $\mathrm{ED}^w(X,Y) \geq \mathrm{ED}(X,Y)$ (by norm.)
$\rightsquigarrow$ Use unweighted $O(kn)$-time ASM as Filter

If $\mathrm{ED}(P, T[\,a\,..\,b\,]) \leq k$, also get cheap alignment
$P \leadsto\!\!\!\!\!\twoheadrightarrow T[\,a\,..\,b\,]$ that matches exactly most of P

All such alignments very similar to each other
But how to exploit this fact?
$\rightsquigarrow$ Cast as shortest path prob in alignment graphs
$\rightsquigarrow$ Copy matched parts from (trivial) align. $P \leadsto\!\!\!\!\!\twoheadrightarrow P$

Comp. bound-to-bound dist for each block [Klein'05];
then combine using fast (min, +)-product [SMAWK]



Searching for $\tilde{O}(kt)$-time Algorithms for Weighted ASM

Weighted Approximate String Matching

Given: T, P, k, oracle to w; Find: (starting pos. of) substr. of T at weighted ED $\le k$ to P.

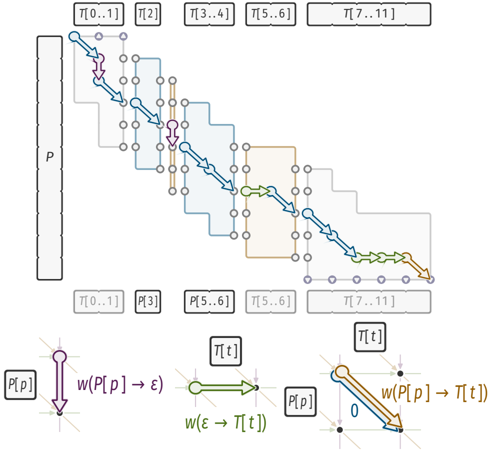
Focus: Obtain starting positions of occurrences

Observation: $\mathrm{ED}^w(X, Y) \ge \mathrm{ED}(X, Y)$ (by norm.)
⇝ Use unweighted $O(kn)$-time ASM as Filter

If $\mathrm{ED}(P, T[a\,..\,b]) \le k$, also get cheap alignment $P \rightsquigarrow\!\!\!\!\gg T[a\,..\,b]$ that matches exactly most of P

All such alignments very similar to each other
But how to exploit this fact?
⇝ Cast as shortest path prob in alignment graphs
⇝ Copy matched parts from (trivial) align. $P \rightsquigarrow\!\!\!\!\gg P$

Comp. bound-to-bound dist for each block [Klein'05];
then combine using fast (min, +)-product [SMAWK]
Precompute all P vs P blocks [Klein'05];
⇝ Verify $\le k$ pos in $\tilde{O}(k^2)$ after $\tilde{O}(kp)$ prep. ⇝ $\tilde{O}(kt)$ total



Weighted ASM: Bottom Lines

Weighted Approximate String Matching

Given: T, P, k, oracle to w; Find: (starting pos. of) substr. of T at weighted ED $\leq k$ to P.

Main Theorem

[ChKoW'25]

Weighted ASM is in time $\tilde{O}(t + t/p \cdot k^4)$ and in $O(k^4)$ $\texttt{PILLAR}$ time for $t < {}^3\!/_2\, p + k$.

Main Theorem

[ChKoW'25]

Weighted ASM is in time $\tilde{O}(kt)$.

Weighted ASM: Bottom Lines

Weighted Approximate String Matching

Given: T, P, k, oracle to w; Find: (starting pos. of) substr. of T at weighted ED $\le k$ to P.

Main Theorem [ChKoW'25]

Weighted ASM is in time $\tilde{O}(t + t/p \cdot k^4)$ and in $O(k^4)$ PILLAR time for $t < {}^3\!/_2\, p + k$.

Main Theorem [ChKoW'25]

Weighted ASM is in time $\tilde{O}(kt)$.

Generalization of PILLAR ASM algorithm to weighted setting "easy".

Weighted ASM: Bottom Lines

Weighted Approximate String Matching

Given: T, P, k, oracle to w; Find: (starting pos. of) substr. of T at weighted ED $\le k$ to P.

Main Theorem [ChKoW'25]

Weighted ASM is in time $\tilde{O}(t + t/p \cdot k^4)$ and in $O(k^4)$ PILLAR time for $t < {}^3/_2\, p + k$.

Main Theorem [ChKoW'25]

Weighted ASM is in time $\tilde{O}(kt)$.

Generalization of PILLAR ASM algorithm to weighted setting “easy”.

But, also interesting non-PILLAR algorithms out there.

Open Problems and Future Directions

- Big Question: Close gap between UB $O(n + k^{3.5})$ / LB $\Omega(n + k^2)$ for ASM.
  - An $\tilde{O}(k^2)$-time `PILLAR` algo would imply optimal algorithms in many other settings (e.g. quantum, compressed, etc.)
  - First step: $\tilde{O}(k^3)$-time `PILLAR` algo
  - Stronger (conditional) LB would require new techniques
  - Similar gap in weighted setting

Open Problems and Future Directions

- Big Question: Close gap between UB $O(n + k^{3.5})$ / LB $\Omega(n + k^2)$ for ASM.
    - An $\tilde{O}(k^2)$-time PILLAR algo would imply optimal algorithms in many other settings (e.g. quantum, compressed, etc.)
    - First step: $\tilde{O}(k^3)$-time PILLAR algo
    - Stronger (conditional) LB would require new techniques
    - Similar gap in weighted setting
- Big Question: Reporting substrings instead of starting positions
    - Alignment graph-based methods seem to fare better
    - First step: $k^{4-\varepsilon}$-time PILLAR algo for ASM
    - First step: $\tilde{O}(kp)$-time for WASM

Open Problems and Future Directions

- Big Question: Close gap between UB $O(n + k^{3.5})$ / LB $\Omega(n + k^2)$ for ASM.
  - An $\tilde{O}(k^2)$-time `PILLAR` algo would imply optimal algorithms in many other settings (e.g. quantum, compressed, etc.)
  - First step: $\tilde{O}(k^3)$-time `PILLAR` algo
  - Stronger (conditional) LB would require new techniques
  - Similar gap in weighted setting
- Big Question: Reporting substrings instead of starting positions
  - Alignment graph-based methods seem to fare better
  - First step: $k^{4-\varepsilon}$-time PILLAR algo for ASM
  - First step: $\tilde{O}(kp)$-time for WASM
- Big Question: Simpler algorithms with similar guarantees
  - Especially almost periodic case highly involved
  - Structural insights currently come with impractical constants

Open Problems and Future Directions

- Big Question: Close gap between UB $O(n + k^{3.5})$ / LB $\Omega(n + k^2)$ for ASM.
  - An $\tilde{O}(k^2)$-time PILLAR algo would imply optimal algorithms in many other settings (e.g. quantum, compressed, etc.)
  - First step: $\tilde{O}(k^3)$-time PILLAR algo
  - Stronger (conditional) LB would require new techniques
  - Similar gap in weighted setting
- Big Question: Reporting substrings instead of starting positions
  - Alignment graph-based methods seem to fare better
  - First step: $k^{4-\varepsilon}$-time PILLAR algo for ASM
  - First step: $\tilde{O}(kp)$-time for WASM
- Big Question: Simpler algorithms with similar guarantees
  - Especially almost periodic case highly involved
  - Structural insights currently come with impractical constants
- Big Question: Generalizing idea behind PILLAR to other problem families, settings

Navigation

Start

Contents

End