Exact and Heuristic Approaches for Solving String Problems from Bioinformatics

Marko Djukanovic

Institute of Logic and Computation
Algorithms and Complexity Group
TU Wien, Vienna, Austria

Supervised by: Guenther Raidl
Co-supervised by: Christian Blum

Financed by: Doctoral programme VGSCO

PhD Defense
May 5th, 2021

Overview

Introduction

Introduction

A *string* is a finite sequence of characters over some (finite) alphabet Σ.

Strings are commonly used as models of

- DNA and RNA molecules, proteins, texts, etc.

Bioinformatics and string problems related:

- detecting similarities between molecules serve for understanding of biological processes (diseases, developmental defects etc.)
- (discrete) optimization problems

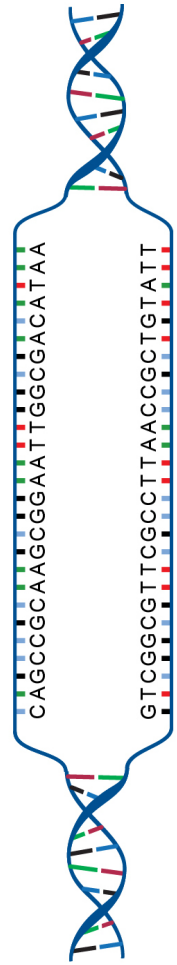


Fig.: a DNA molecule representing string `CAGCCCG...`

String problems

- Measures of similarity between (different molecular) structures
- Defining such similarity measures important in various fields
  - molecular biology
  - text processing
  - information retrieval
  - gene recognition, pattern matching
  - data compression, sequence analysis
  - file plagiarism check, etc.
- Algorithms that solve these problems appear as:
  - the basis of version control system GIT
  - the basis of Unix command DIFF

Most Important String Problems

In the literature there are dozens of such problems.

A *subsequence* of string s is any sequence of characters obtained by deleting zero or more characters from s.

Most Important String Problems

In the literature there are dozens of such problems.

A *subsequence* of string s is any sequence of characters obtained by deleting zero or more characters from s.

Longest Common Subsequence (LCS) Problem:

- Input: set of strings $S = \{s_1, \ldots, s_m\}$, $m \in \mathbb{N}$, alphabet Σ
- Objective: find a *subsequence* of *maximum* length that is *common* for all strings from S
- $\mathcal{NP}$–hard problem if m arbitrary
- Example: $|S| = 2$, $S = \{\texttt{abca}\texttt{bcd}\texttt{a}, \texttt{a}\texttt{cc}\texttt{bc}\texttt{caa}\}$, $\Sigma = \{a, b, c, d\}$. A longest common subsequence (LCS): $\texttt{abcaa}$

LCS problem: Literature

- The basic $m = 2$ case has been solved over last 60 years, dozens of approaches (within 70s, 80s and 90s)
- Around 2000s, with development of powerful heuristic algorithms, the case of arbitrary m becomes the main focus:

2001	Huang et al.: Greedy Heuristic (GH)
2008	Easton and Singireddy: Large neighborhood search (makes internal use of specific branching)
2009	Shyu and Tsai: ACO
2009	Blum et al.: Beam Search (BS-Blum)
2010	Wang et al.: A*-based heuristic (BS-Wang)
2012	Mousavi and Tabataba: a novel BS approach (BS-H)
2013	Mousavi and Tabataba: A Hyper-heuristic (HH) approach + new heuristic guidance (BS-Pow)
2019	Islam et al.: Chemical Reactive Optimization

LCS: Exact Approaches from the Literature

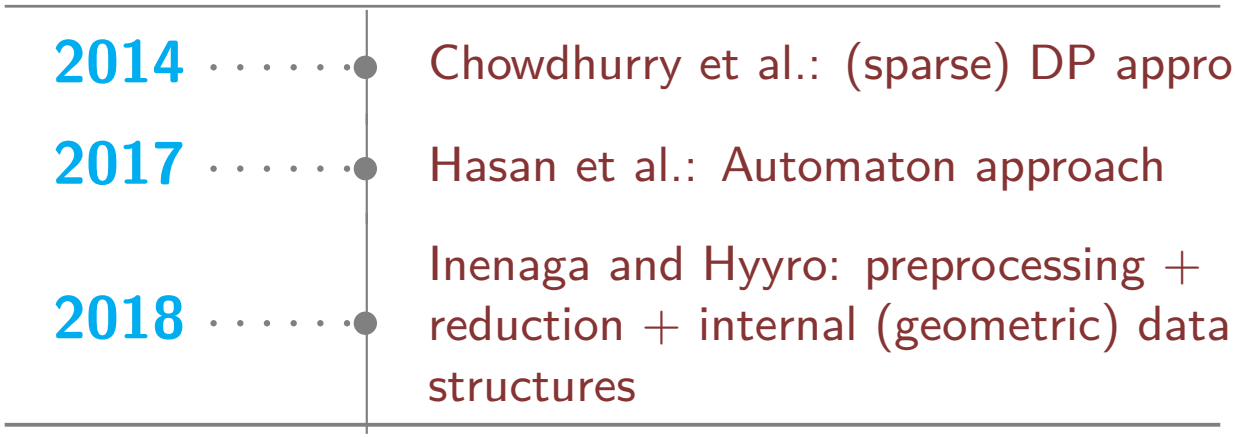
2000	Bergroth et al.: Advanced DP approach
2009	Lee and Gupta: two ILP models
2011	Wang et al.: the parallel Quick_DP
2013	Yang et al.: Pro–MLCS anytime algorithms – iterated widening beam search
2016	Chen et al.: the parallel FAST_LCS
2016	Li et al.: the parallel TOP_MLCS
2017	Peng and Wang: the parallel Leveled-DAG

Longest Common Palindromic Subsequence (LCPS) Problem

- Input: a set of input strings $S = \{s_1, \ldots, s_m\}$, $m \in \mathbb{N}$
- Objective: find a subsequence of maximum length that is common for all strings from S and is also a palindrome
- Introduced by Chowdhurry et al. (2014)
- Example: S={aabbaca, adcbba}, LCPS is abba.
- Practical relevance:
  - Detecting cancer cells
  - Instability in gene functioning $\Longleftrightarrow$ presence of genes as palindromic sequences
- Complexity:
  - $\mathcal{NP}$–hard if m arbitrary

LCPS Approaches

- Studied for $m = 2$



- To the best of our knowledge, no approach dealing with arbitrary m

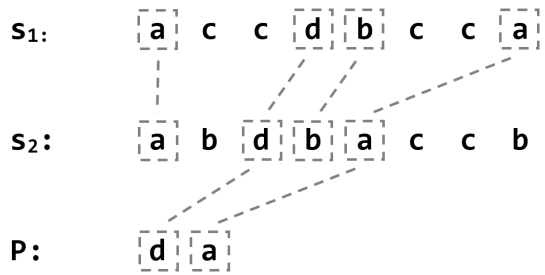
Constrained Longest Common Subsequence (CLCS) Problem

- Input: a set of strings $S = \{s_1, \ldots, s_m\}, m \in \mathbb{N}$, alphabet Σ, and a pattern string P.
- Objective: find a *subsequence* of maximum length that is common to all strings from S and has pattern P as its subsequence.

Constrained Longest Common Subsequence (CLCS) Problem

- Input: a set of strings $S = \{s_1, \ldots, s_m\}, m \in \mathbb{N}$, alphabet Σ, and a pattern string P.
- Objective: find a *subsequence* of maximum length that is common to all strings from S and has pattern P as its subsequence.

  Example:



Practical relevance: identifying homology between biological sequences which have a specific or putative structure in common:

- `Rnase`, `Kinease`, `Protease`, `Globulin`, etc.

Literature Overview

- Introduced by Chin and Tsai (2003), the $m = 2$ case
- *Polynomially* solvable by dynamic programming (DP), a few sparse DP approaches:

2003	Chin and Tsai: basic DP
2005	Arslan and Eğecioğlu: multiple 3D matrices
2007	Deorowicz et al.: a sparse DP
2008	Iliopoulos and Rahman: bounded heap, veB trees
2018	Hung et al.: a diagonal–based approach, idea of partial solutions and extensions

- $\mathcal{NP}$-hard if m arbitrary
- Approximation algorithm by Gotthilf et al. (2008): appr. factor of $\sqrt{l_{min} \cdot |\Sigma|}$ where $l_{min} = \min\{|s_1|, \ldots, |s_m|\}$

The Longest Arc-Preserving Common Subsequence (LAPCS) Problem

- Arc-annotated strings:
  - models for DNA/RNA molecules
  - (s, A), s is string and $A \subseteq (1, \ldots, |s|) \times (1, \ldots, |s|)$ set of arcs (pairs of positions)

  $(s = \text{AUGGACGCGU}, A = \{(1, 10), (4, 6), (7, 8)\})$

- Input: two arc–annotated strings $(s_1, A_1), (s_2, A_2)$
- Objective: find a subsequence of maximum length that is common to both strings and preserves arcs between characters.
- Example:

LAPCS

- Introduced by Evans (1999)
- General case: no restriction between positions of two different arcs in each of the input strings $\Longrightarrow \mathcal{NP}$–hard
- Practical relevance: similarity between arc-annotated seqs
- DP–based heuristic, Jiang et al. (2004)
- ILP model and a hybrid EA (HYB-EA) based on solution merging technique, Blum and Blesa (2017)

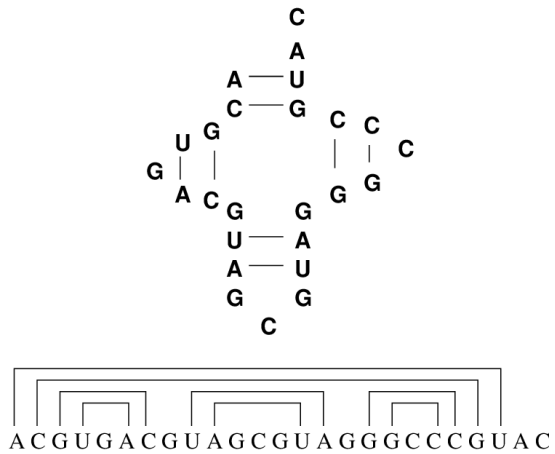


Fig.: Example of an RNA molecule and its corresponding arc-annotation shape.

Repetition–Free Longest Common Subsequence (RFLCS) Problem

- Input: s_1 and s_2, and alphabet Σ
- Objective: find a *subsequence* of maximum length that is common to both strings and has *no character* occurring *more than once* in the subsequence
- Example: $s_1 = \texttt{dabcbacbab}, s_2 = \texttt{abbcccbad}$; RFLCS: $\texttt{cba}$.
- Introduced by Adi et al. (2010)
- Practical relevance:
  - deals with the molecules that come from different origins
  - the gene rearrangement domain
- Complexity: $\mathcal{APX}$–hard, which implies $\mathcal{NP}$–hardness, Adi et al. (2010)

Literature Approaches for RFLCS Problem

2010	Adi et al.: 2-approximate algorithm, ILP model
2013	Blum et al.: a hybrid Beam-ACO
2013	Castelli et al.: an evolutionary algorithm
2016	Blum and Blesa: Construct, Merge, Solve and Adapt (CMSA)
2018	Blum and Blesa: hybrid of probabilistic Beam Search and CMSA – HYB-CMSA

Longest Common Square Subsequence (LCSqS) Problem

String s is a square string iff $\exists\ s'$ such that $s = s' \cdot s' = (s')^2$ (abab is a square string)

- Input: a set of strings $S = \{s_1, \ldots, s_m\}, m \in \mathbb{N}$
- Objective: find a subsequence of maximum length that is common to all strings from S and is a square string
- Example: $s_1 =$ dabcdab, $s_2 =$ adaccdad $\Longrightarrow$ LCSqS: dada
- Introduced by Inoue et al. (2018): DP and two sparse approaches ($m = 2$), but no approach for $m > 2$
- Complexity:
  - $\mathcal{NP}$–hard if m arbitrary
- Practical relevance: a measure of similarity between disjunctive parts of each of molecules (internal similarity)

Contributions

Contributions Concerning LCS Problem

- A paper published in proceedings of *LOD 2019* – the 5th International Conference on Machine Learning, Optimization, and Data Science (2019):
  - Generalized Beam Search Framework
  - A novel expected length calculation heuristic
- A paper published in *Applied Soft Computing* (2020):
  - Two hybrids A*+BS and A*+ACS
- A paper submitted to *Mathematics* (2021):
  - Solving LCS on Non-uniform Distributions
  - Novel search guidance

Contributions Concerning CLCS Problem

- A paper published in proceedings of *OPTIMA 2020* – the 11th International Conference Optimization and Applications (2020):
  - Consider the arbitrary m case
  - Greedy method, Beam search (BS), various search guidances
  - A* search
- A paper published in *Information Processing Letters* (2020):
  - Compreh. comparisons between exact approaches for $m = 2$
- A paper submitted to *Applied Soft Computing* (2021):
  - Considered the generalized CLCS problem with arbitrary many pattern strings
  - Hybrid of a restricted BS and Variable Neighborhood Search (VNS)

Contributions [cont'd]

Contributions Concerning LCPS problem:

- A paper published in proceedings of *LION 12* – the 12th International Conference on Learning and Intelligent Optimization (2018):
  - Consider the arbitrary m case
  - Greedy method, BS
  - A* search and hybrid A*+BS
- A paper published in *Computers & Operations Research* (2020):
  - The expected length calculation heuristic for LCPS
  - Developed a novel hybrid A*+ACS

Contributions Concerning RFLCS / LAPCS problems:

- A paper published in *Computers & Operations Research* (2021):
  - Transformation to Max-Clique instances
  - Problem-specific graph reduction

Contributions [cont'd]

Contributions Concerning LCSqS problem:

- A paper published in proceedings of EUROCAST 2019 – the 17th International Conference on Computer Aided Systems Theory (2019):
  - Consider the arbitrary m case
  - Randomized Local Search
  - Hybrid BS & VNS

Theses:

- One bachelor thesis on the RFLCS problem (2020)
- One master thesis on the CLCS problem (2020)

M. Djukanovic, Günther R. Raidl, and C. Blum. Finding Longest Common Subsequences: New Anytime A* Search Results. ASOC, 2020.

Search Space for LCS Problem

- $s[x, |s|]$: suffix (substring) of s which starts at position x and includes the last letter of s

Search Space for LCS Problem

- $s[x, |s|]$: suffix (substring) of s which starts at position x and includes the last letter of s
- For $p^{\mathrm{L}} \in \mathbb{N}^m$, we define an LCS subproblem as
  - a set of suffixes $S[p^{\mathrm{L}}] = S[(p_1^{\mathrm{L}}, \ldots, p_m^{\mathrm{L}})] := \left\{ s_i[p_i^{\mathrm{L}}, |s_i|] \mid i = 1, \ldots, m \right\}$: the (remaining) parts of strings from S w.r.t. p^{L}

Search Space for LCS Problem

- $s[x, |s|]$: suffix (substring) of s which starts at position x and includes the last letter of s
- For $p^{\mathrm{L}} \in \mathbb{N}^m$, we define an LCS subproblem as
  - a set of suffixes
    $S[p^{\mathrm{L}}] = S[(p_1^{\mathrm{L}}, \ldots, p_m^{\mathrm{L}})] := \{s_i[p_i^{\mathrm{L}}, |s_i|] \mid i = 1, \ldots, m\}$:
    the (remaining) parts of strings from S w.r.t. p^{L}
- **Example:** $S = \{\texttt{abbbbcbbba}, \texttt{accbcbcbbb}, \texttt{cbbacbcba}\}$ and $p^{\mathrm{L}} = (2, 5, 3)$:
  $S[(2, 5, 3)] = \{\texttt{abbbbcbbba}, \texttt{accbcbcbbb}, \texttt{cbbacbcba}\}$

Search Space for LCS Problem

- $s[x, |s|]$: suffix (substring) of s which starts at position x and includes the last letter of s
- For $p^{\mathrm{L}} \in \mathbb{N}^m$, we define an LCS subproblem as
  - a set of suffixes
    $S[p^{\mathrm{L}}] = S[(p_1^{\mathrm{L}}, \ldots, p_m^{\mathrm{L}})] := \{s_i[p_i^{\mathrm{L}}, |s_i|] \mid i = 1, \ldots, m\}$:
    the (remaining) parts of strings from S w.r.t. p^{L}
- **Example:** $S = \{\texttt{abbbbcbbba}, \texttt{accbcbcbbb}, \texttt{cbbacbcba}\}$ and $p^{\mathrm{L}} = (2, 5, 3)$:
  $S[(2, 5, 3)] = \{\texttt{abbbbcbbba}, \texttt{accbcbcbbb}, \texttt{cbbacbcba}\}$
- String s^p is *a partial solution* of S iff it is a common subsequence for all strings from S:
  - only *meaningful* partial solutions considered

General Search Framework for LCS Problem (through an example)

$$s^p = \varepsilon,\ p^{\mathrm{L}} = (1, 1, 1)$$

$$\boxed{\{\mathtt{abbbbcb}, \mathtt{accbcbc}, \mathtt{bbacca}\}}$$

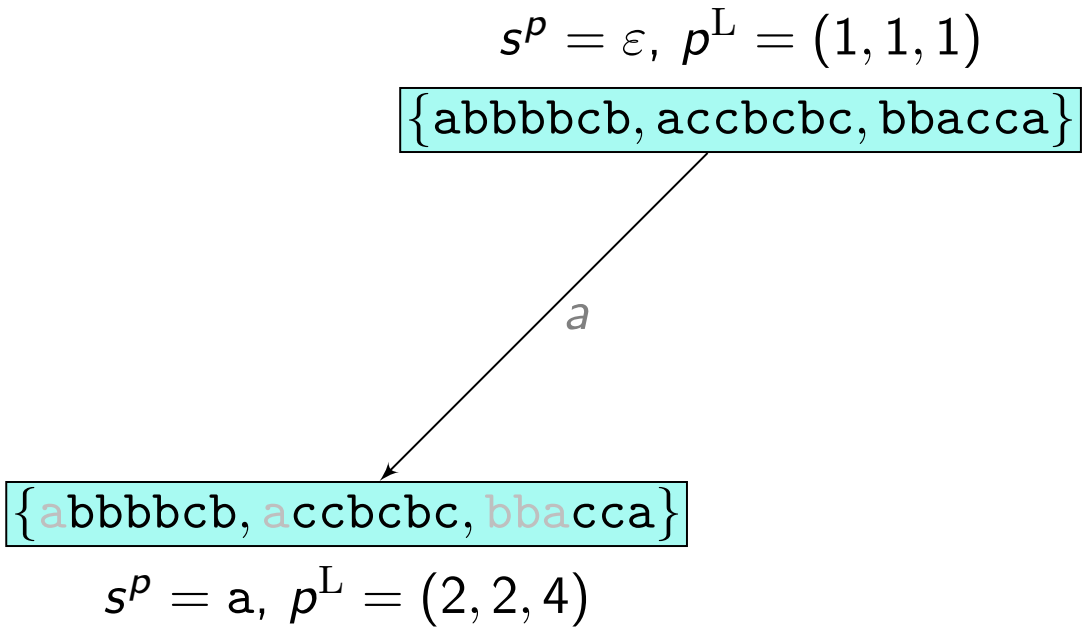
General Search Framework for LCS Problem (through an example)

$s^p = \varepsilon,\ p^{\mathrm{L}} = (1, 1, 1)$

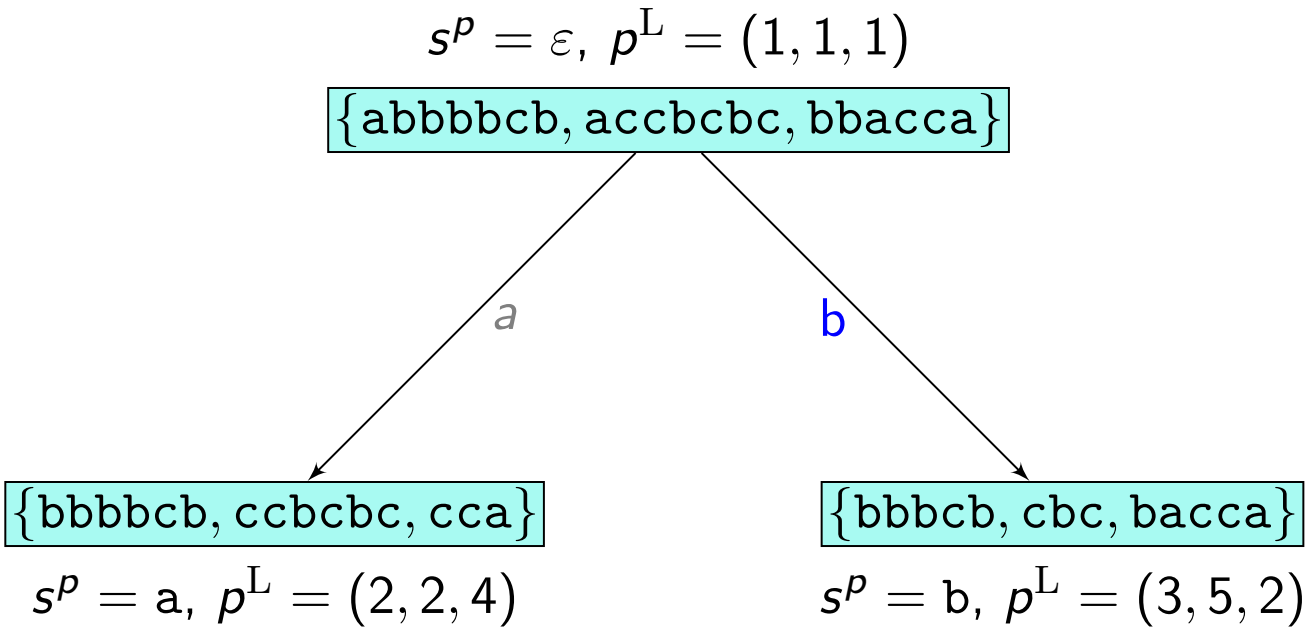
$\{\underline{\mathtt{a}}\mathtt{bbbbcb}, \underline{\mathtt{a}}\mathtt{ccbcbc}, \mathtt{bb}\underline{\mathtt{a}}\mathtt{cca}\}$

$p^{\mathrm{L}}_{3,a} = 3$

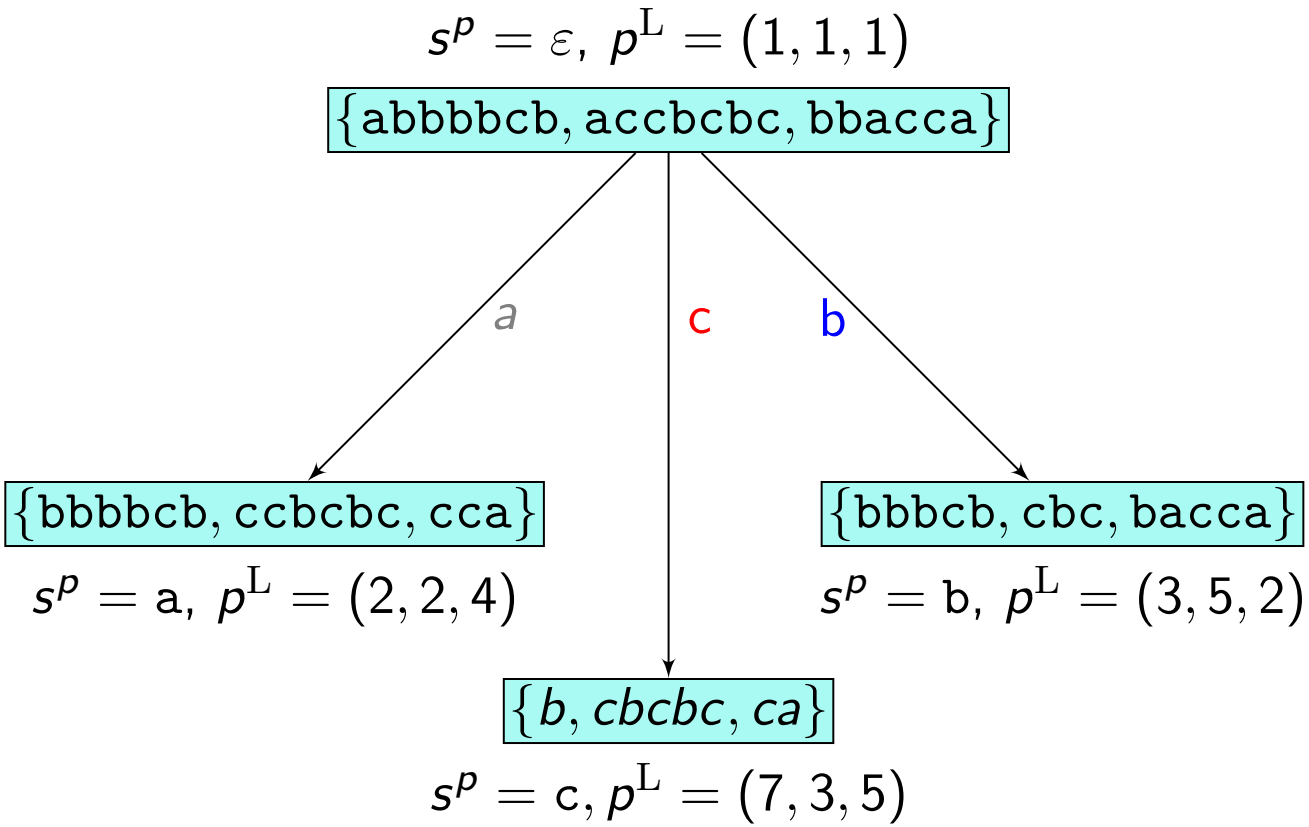
General Search Framework for LCS Problem (through an example)



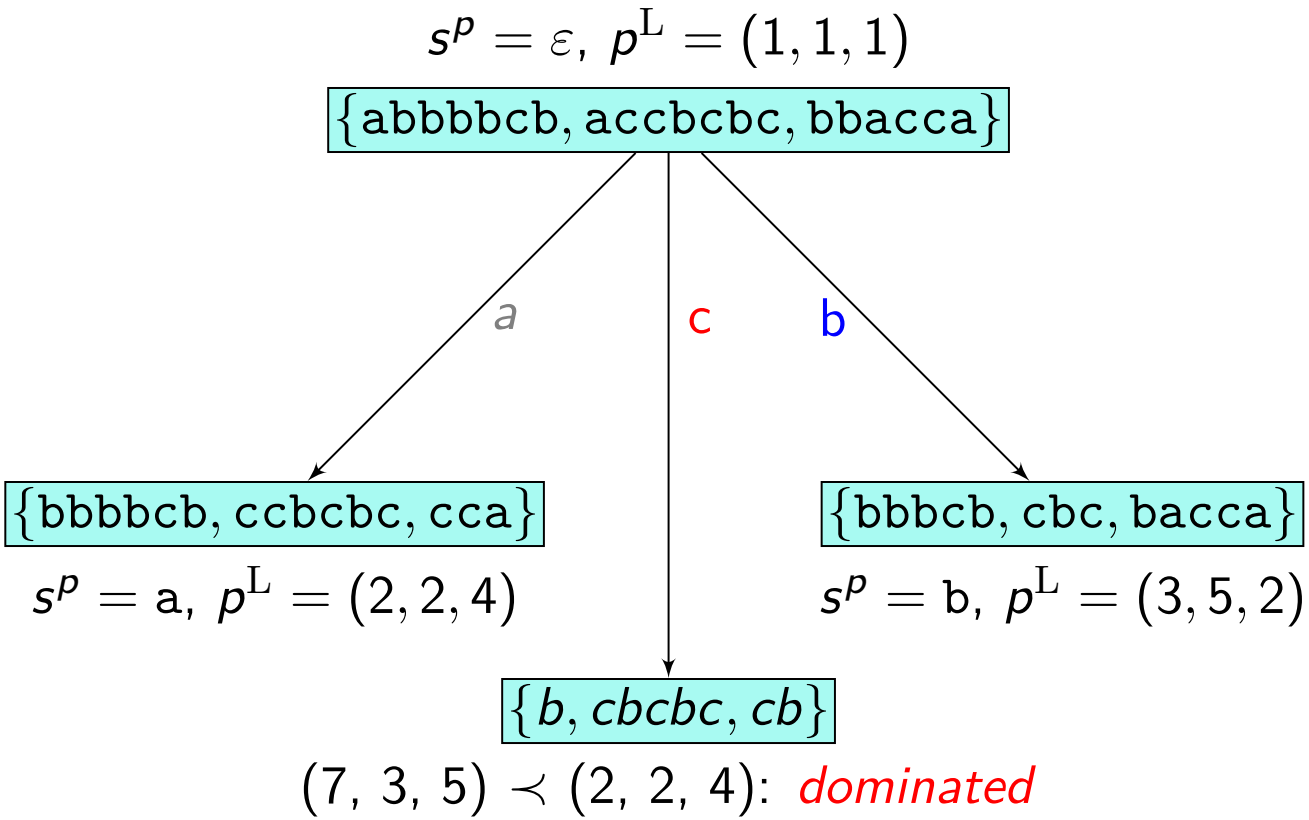
General Search Framework for LCS Problem (through an example)



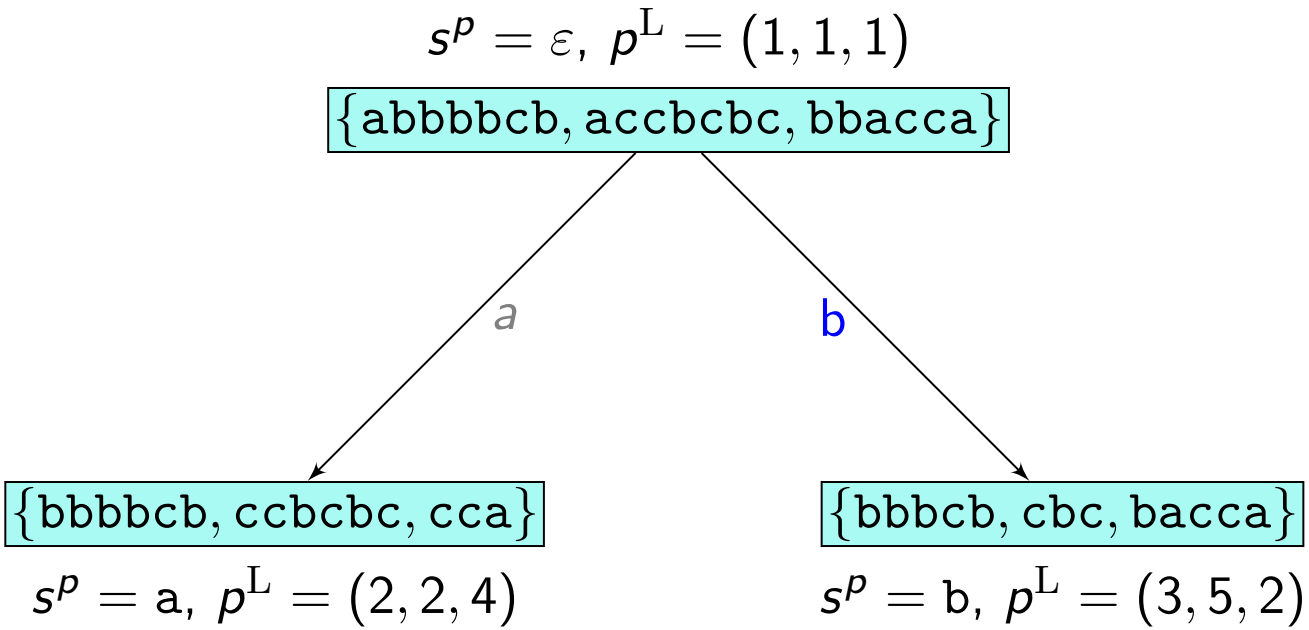
General Search Framework for LCS Problem (through an example)



General Search Framework for LCS Problem (through an example)



General Search Framework for LCS Problem (through an example)



General Search Framework for LCS Problem

- Node v:=$(p^{\mathrm{L},v}, l^v)$:
  - $S[p^{\mathrm{L},v}]$: remaining substrings (suffixes) relevant to extend the partial solution
  - l^v: the length of the corresponding partial solution
- $r = ((1, \ldots, 1), 0)$ is *root* node
- Expansion of v:
  - determine non-dominated feasible letters for $S[p^{\mathrm{L},v}]$
  - extend a partial solution of v by a letter in all possible ways
  - *complete nodes* = non-expanded nodes
- Evaluation of v by:
  - upper bounds: an occurrence–based (UB_1) and a DP–based UBs for LCS (UB_2)
  - different heuristics (not providing bounds)

Beam Search (BS)

- Works in limited breadth-first-search (BFS) manner
- Instead expanding all nodes at each level, BS only expands (up to) $\beta > 0$ best nodes (a heuristic decision)

Beam Search (BS)

- Works in limited breadth-first-search (BFS) manner
- Instead expanding all nodes at each level, BS only expands (up to) $\beta > 0$ best nodes (a heuristic decision)

2001	Greedy criterion
2009	Rank-based heuristic: Rank(v)
2012	Probability–based heuristic: $\mathrm{H}(v)$
2013	Power heuristic: Pow(v)
2019	Expected length heuristic: $\mathrm{Ex}(v)$: LOD2019

Expected Length Calculation Heuristic: Construction

$\mathrm{Ex}(\cdot)$: approximates the expected length of an LCS on random strings

- Assumptions:
  - input strings are random and independent of each other
  - E_k^s: an event that the sequence s of length k appears as a subsequence of all strings from S
  - the probability that E_k^s occurs is independent of the probabilities that event $E_k^{s'}$ occurs, $(\forall s')\ s' \neq s$
- Make use of probability matrix P developed by Mousavi and Tabataba (2012):
  - $P[k, l]$: stores the probability that a random string of length k is a subsequence of any string of length l (a DP recursion)

Exact Approaches for the LCS Problem: A* Search

- Hart et al. (1968)
- Graph traversal and path search algorithm
- Best-first-search technique: the most promising nodes expanded first
- Informed search – nodes prioritized according to $f(v) = g(v) + h(v)$, $v \in V(G)$, where
  - $g(v)$: length of the longest path from root node to v
  - $h(v)$: estimated length of the path from v to a goal node (dual bound)
- A node with the highest f-value expanded first

A* Search: Set up

- $g(v) := l^v$ and $h(v) := \mathrm{UB}(v) = \min\{\mathrm{UB}_1(v), \mathrm{UB}_2(v)\}$
- Complete nodes: goal nodes (no successors)
- Initialize: priority queue $Q = \{r\}$ // set od not-yet-expanded nodes
- Repeat:
  - pop node $u \in Q$ with maximum f value
  - if u is a goal node $\Longrightarrow$ terminate with a proven optimum
  - expand u: determine successor nodes
  - update Q

A*: limitations

- A* search solving:
  - often weak primal solutions upon termination
  - a proven dual bound obtained

To improve anytime behaviour of exact algorithms $\Longrightarrow$ anytime algorithms developed (1980s):

- Trade–off between:
  - keeping chances to prove optimality
  - providing reasonable solutions in almost any times
- keep improving primal solutions over time
- if enough resources ensured, optimality is proven

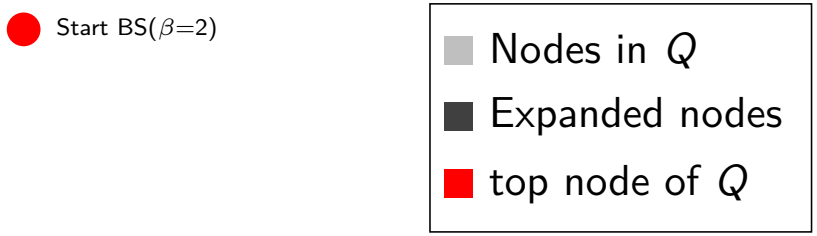
Anytime algorithms

- Recently developed
  - Anytime pack search (APS) and
  - Anytime column search (ACS) state-of-the-art on many different fields.
- Idea of constructing our anytime algorithms:
  - A* framework used to ensure completeness
  - Perform heuristic techniques that are interleaved within $\delta > 0$ classical A* iterations

The A*+BS Hybrid

Heuristic technique we use:

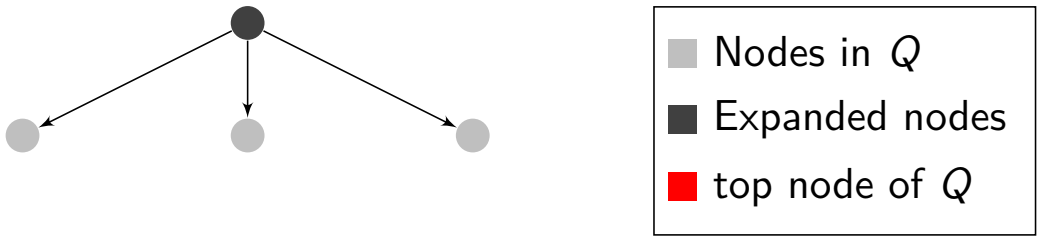
- BS with beam width β: initial beam takes the top node of Q



The A*+BS Hybrid

Heuristic technique we use:

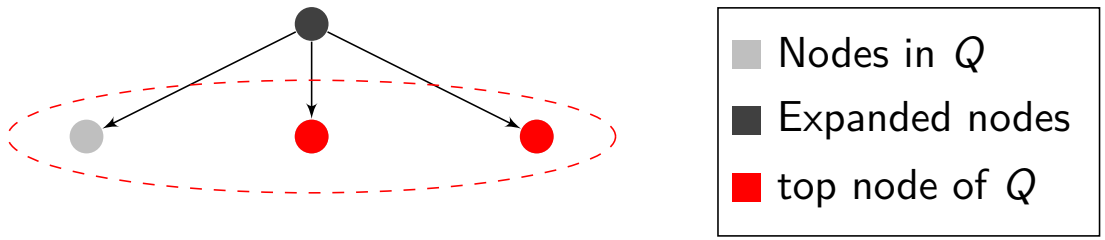
- BS with beam width β: initial beam takes the top node of Q



The A*+BS Hybrid

Heuristic technique we use:

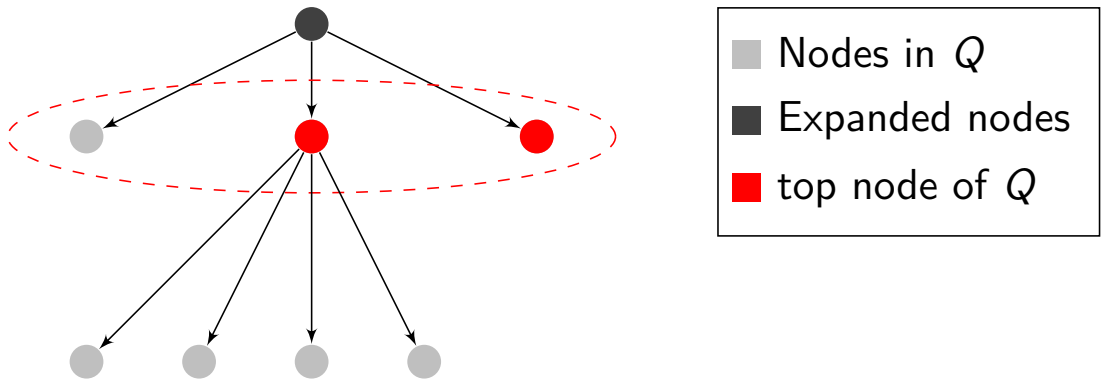
- BS with beam width β: initial beam takes the top node of Q



The A*+BS Hybrid

Heuristic technique we use:

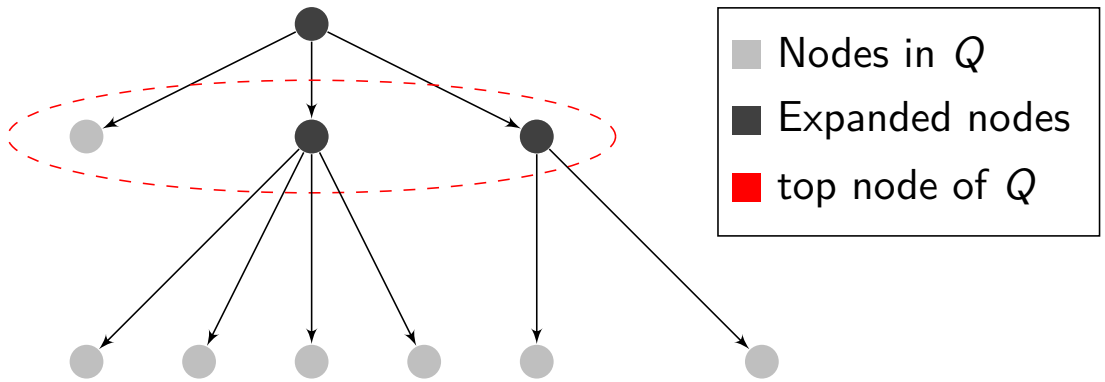
- BS with beam width β: initial beam takes the top node of Q



The A*+BS Hybrid

Heuristic technique we use:

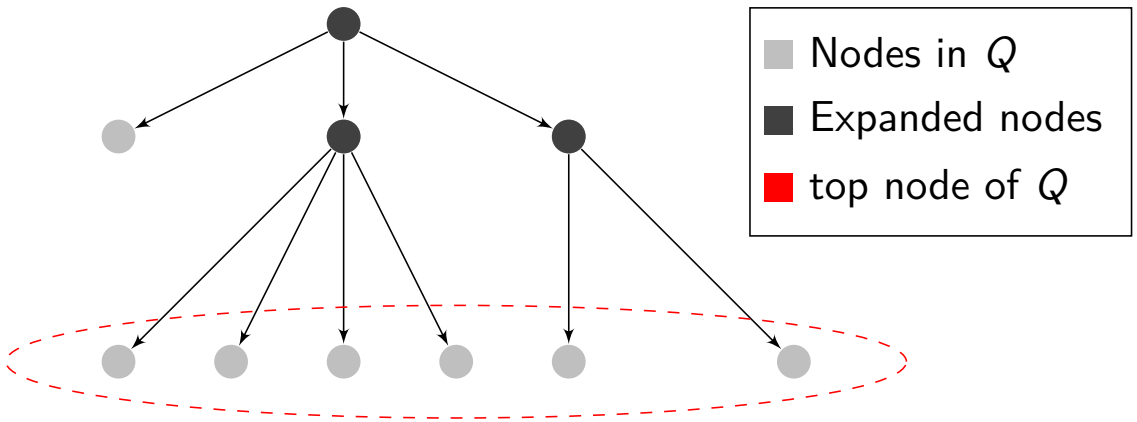
- BS with beam width β: initial beam takes the top node of Q



The A*+BS Hybrid

Heuristic technique we use:

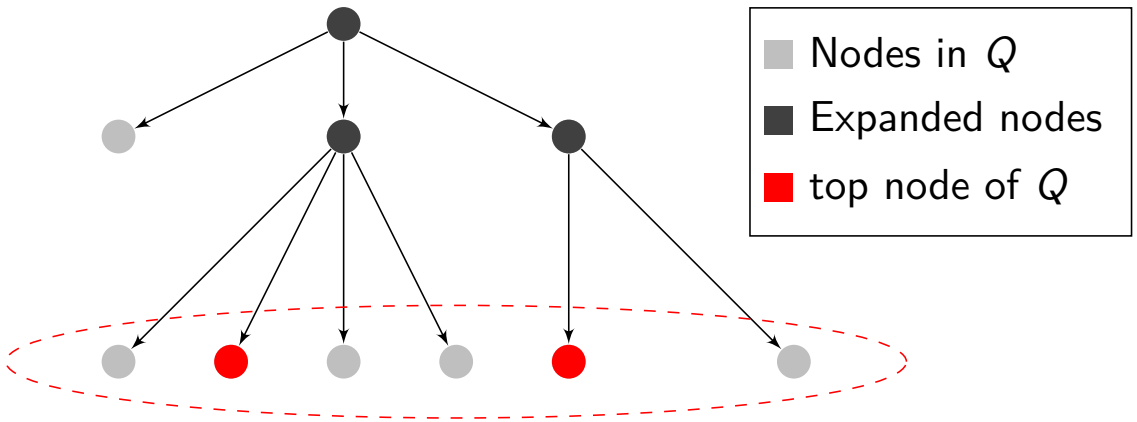
- BS with beam width β: initial beam takes the top node of Q



The A*+BS Hybrid

Heuristic technique we use:

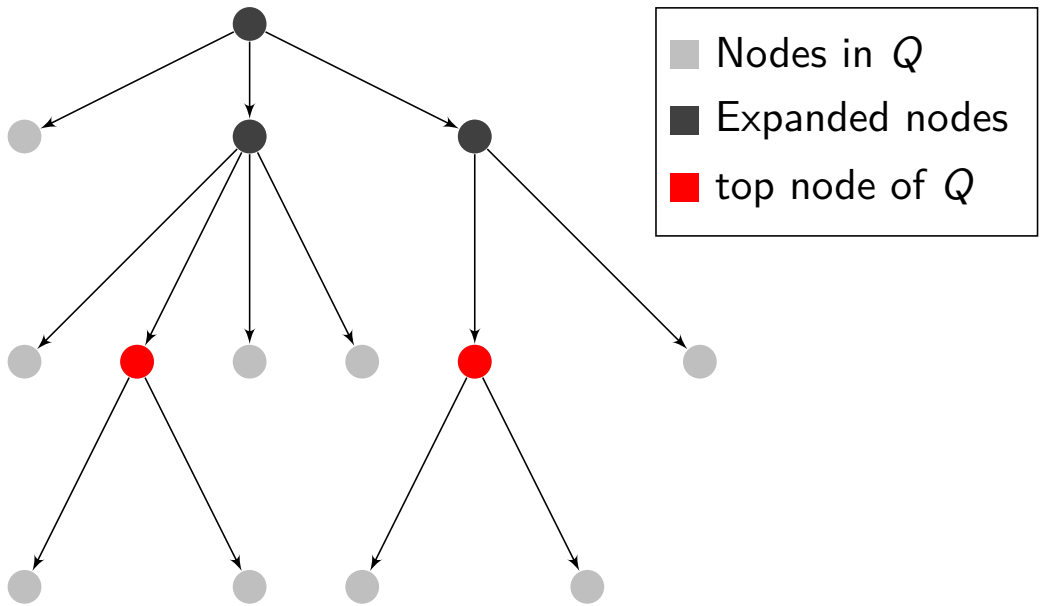
- BS with beam width β: initial beam takes the top node of Q



The A*+BS Hybrid

Heuristic technique we use:

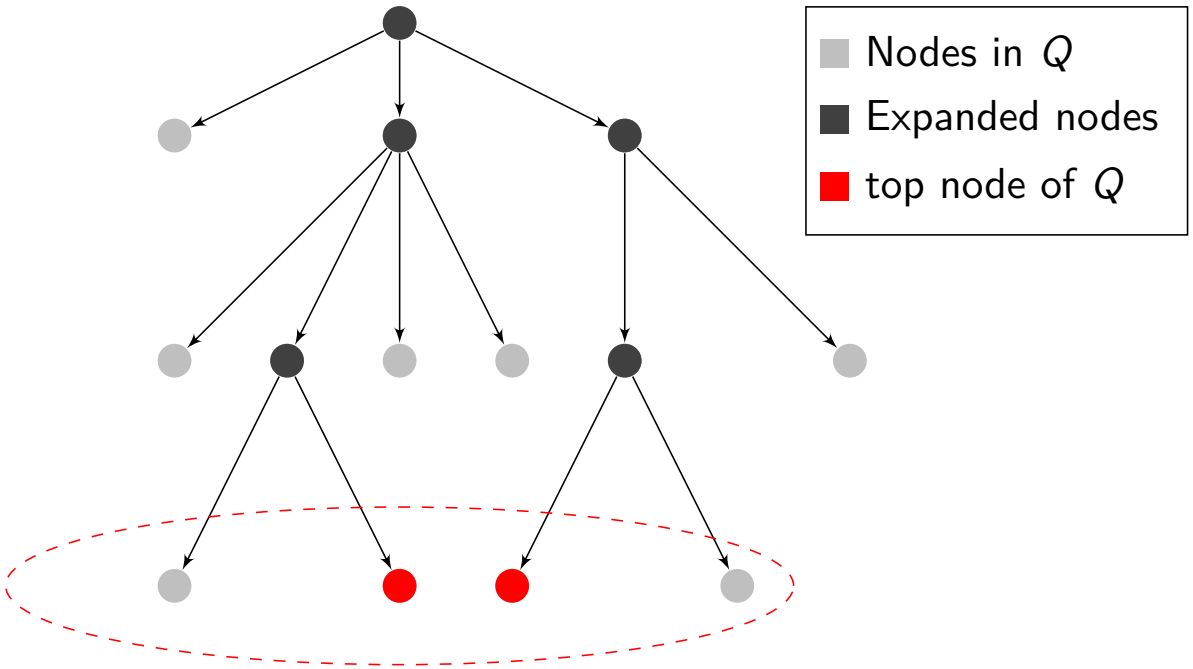
- BS with beam width β: initial beam takes the top node of Q



The A*+BS Hybrid

Heuristic technique we use:

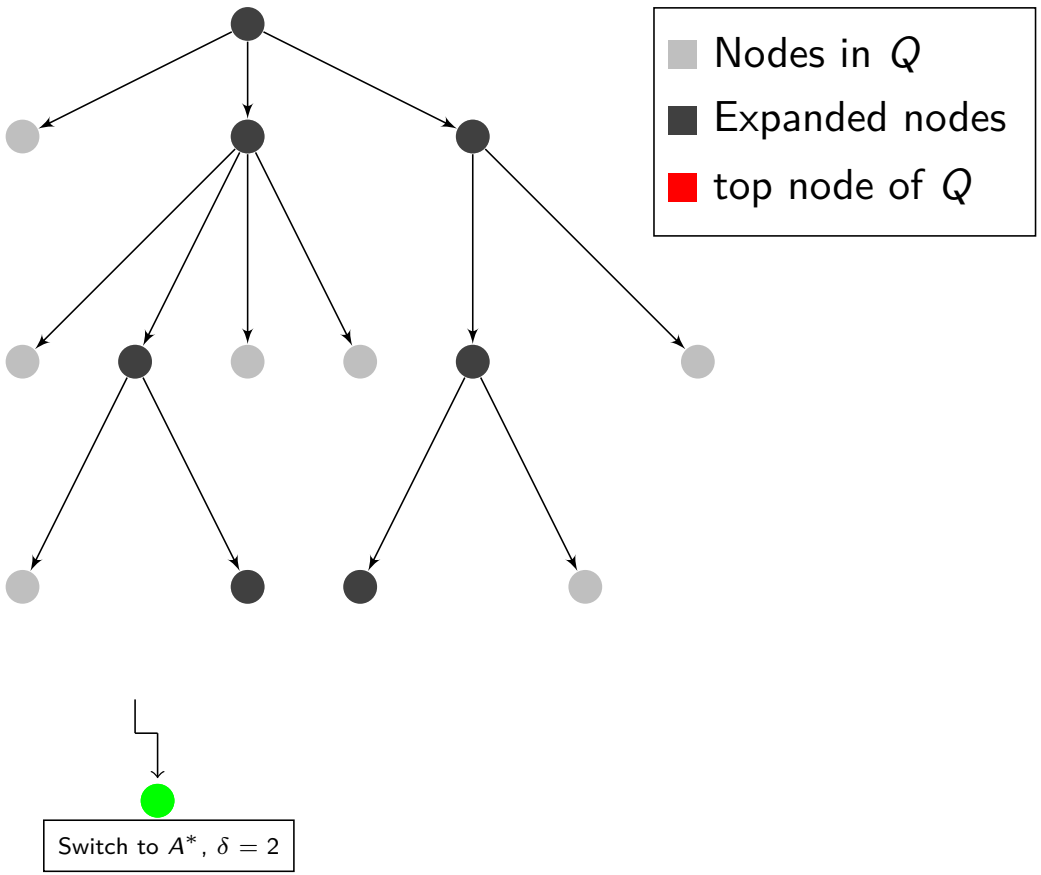
- BS with beam width β: initial beam takes the top node of Q



The A*+BS Hybrid

Heuristic technique we use:

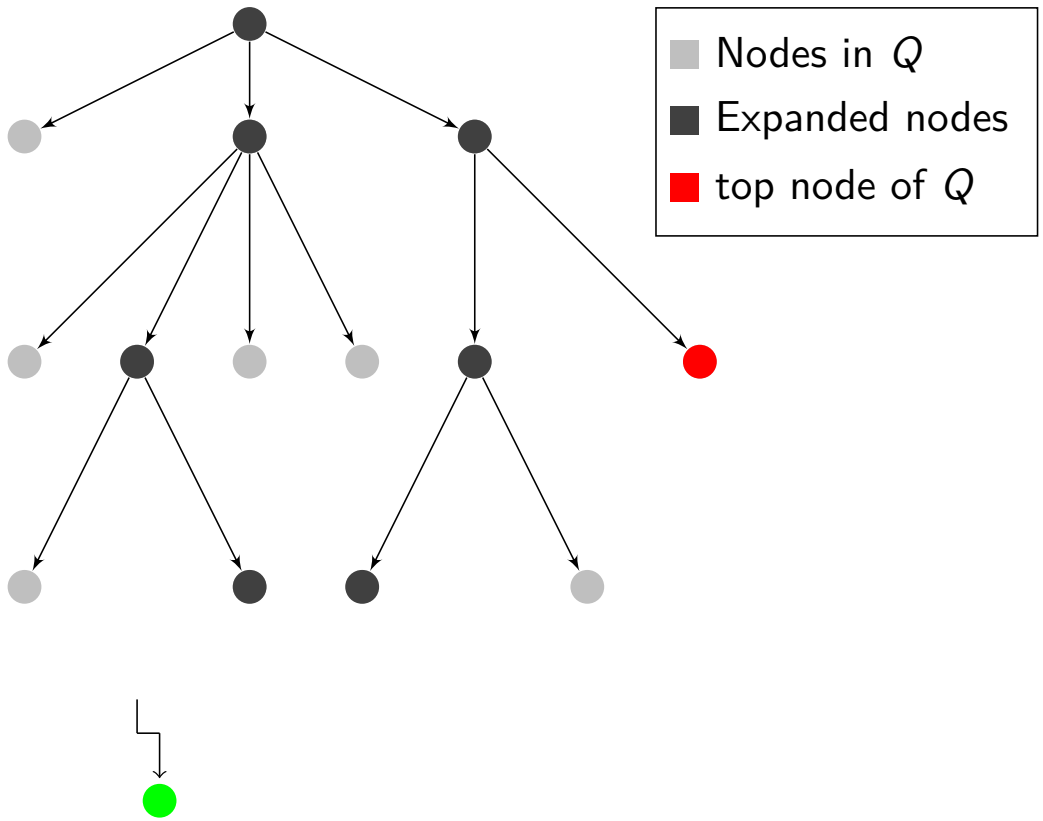
- BS with beam width β: initial beam takes the top node of Q



The A*+BS Hybrid

Heuristic technique we use:

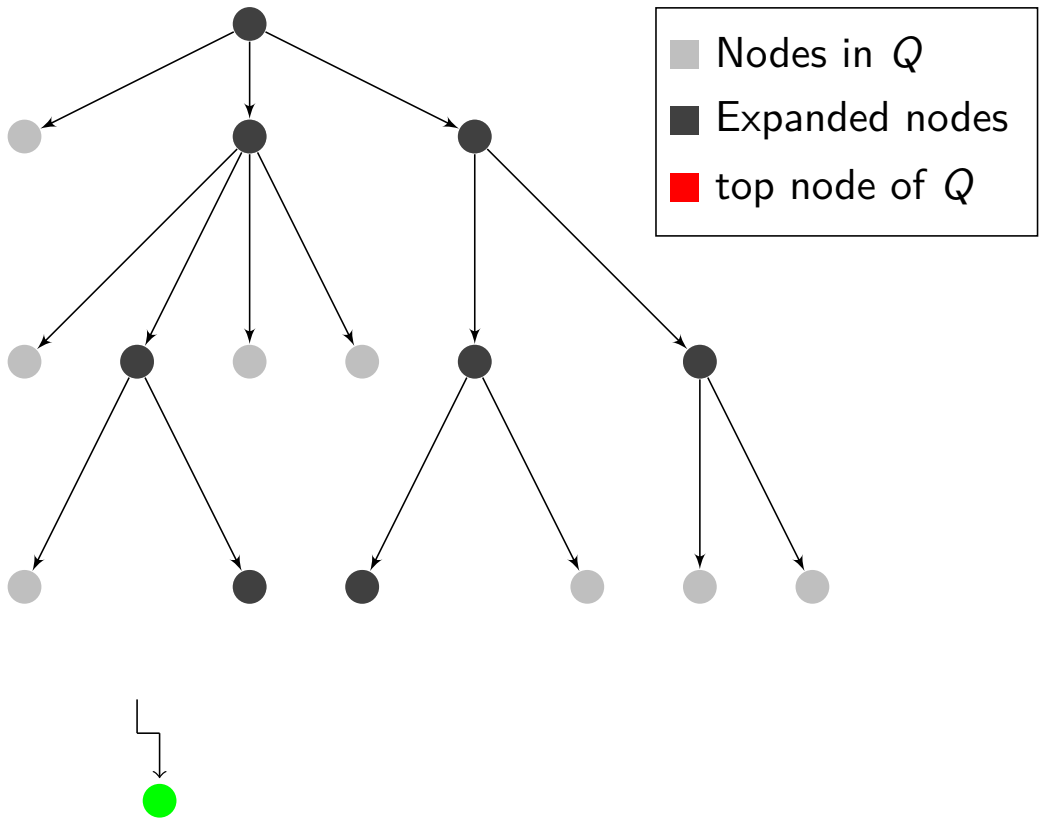
- BS with beam width β: initial beam takes the top node of Q



The A*+BS Hybrid

Heuristic technique we use:

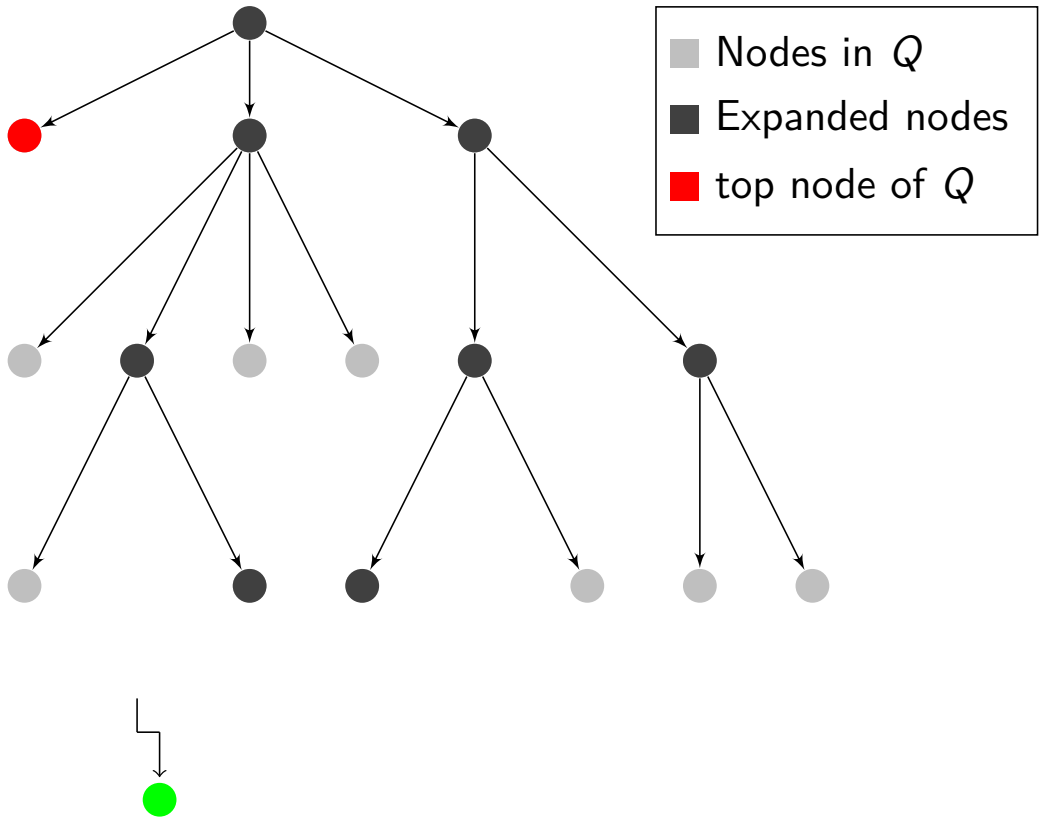
- BS with beam width β: initial beam takes the top node of Q



The A*+BS Hybrid

Heuristic technique we use:

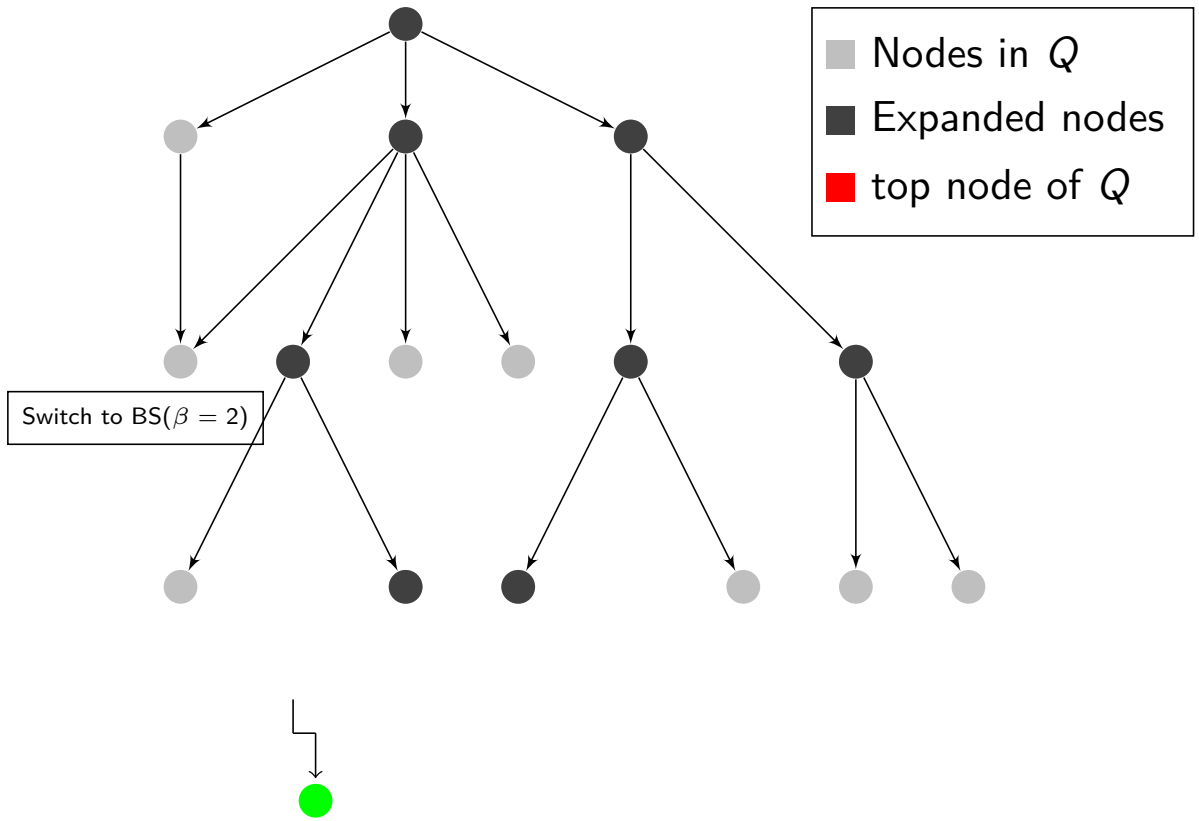
- BS with beam width β: initial beam takes the top node of Q



The A*+BS Hybrid

Heuristic technique we use:

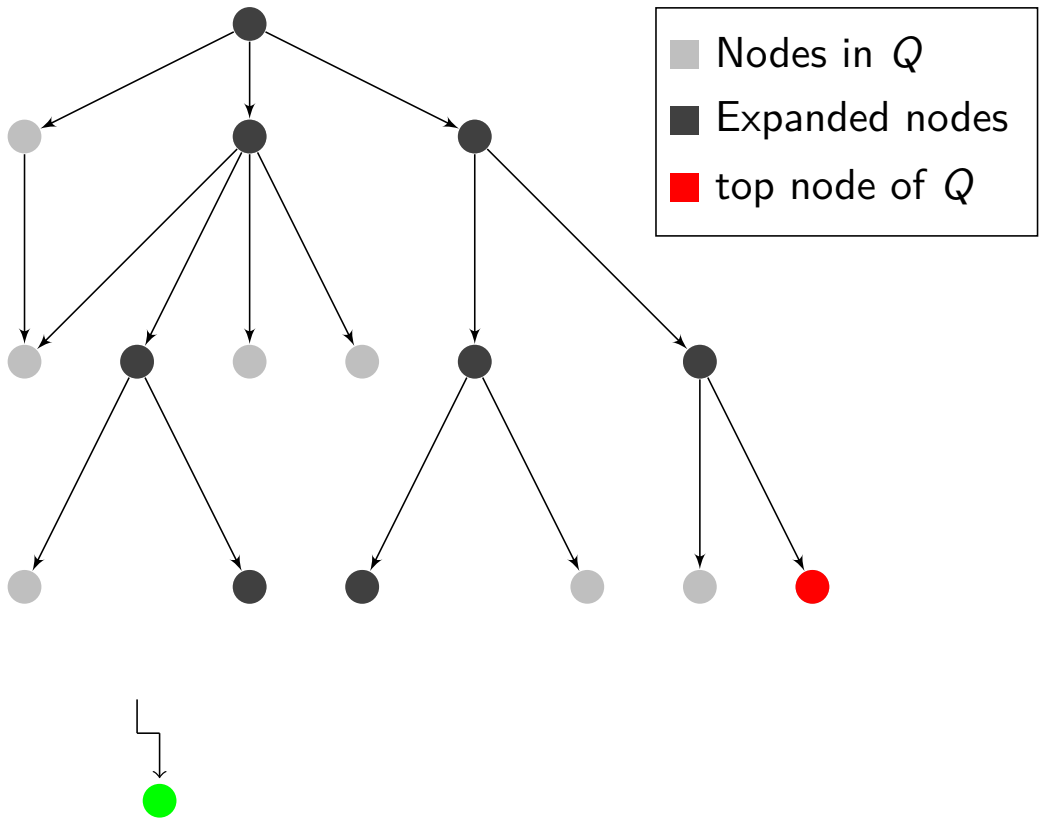
- BS with beam width β: initial beam takes the top node of Q



The A*+BS Hybrid

Heuristic technique we use:

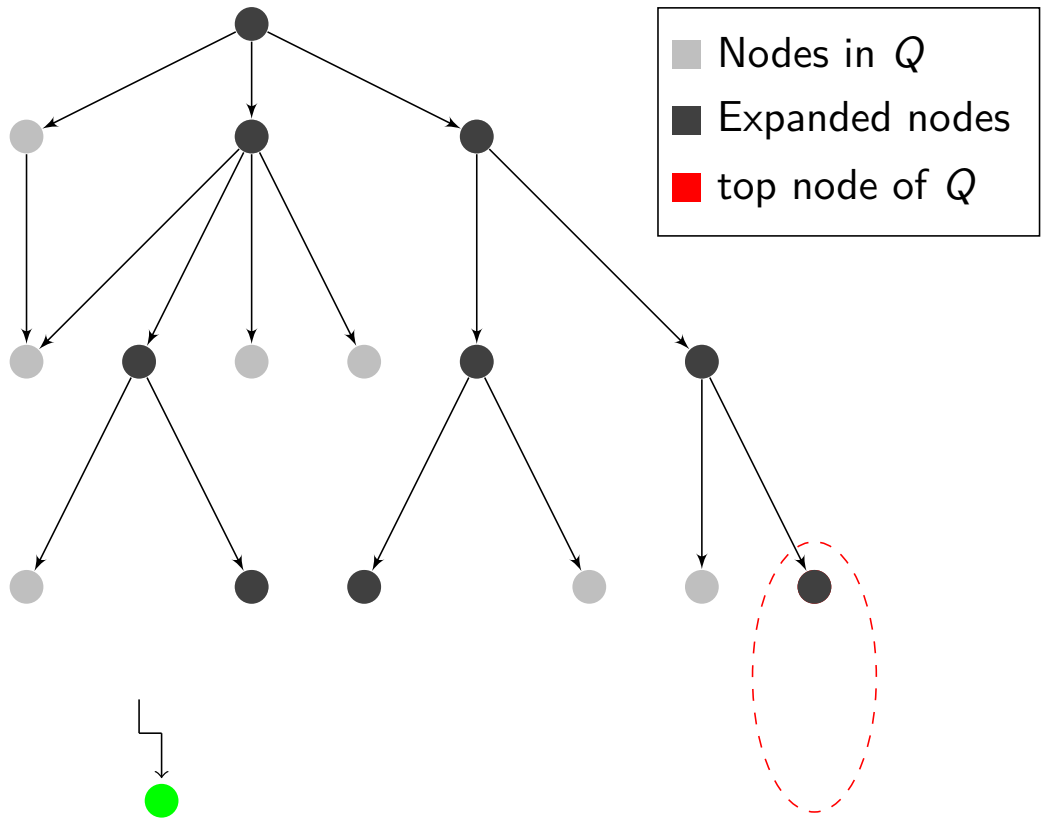
- BS with beam width β: initial beam takes the top node of Q



The A*+BS Hybrid

Heuristic technique we use:

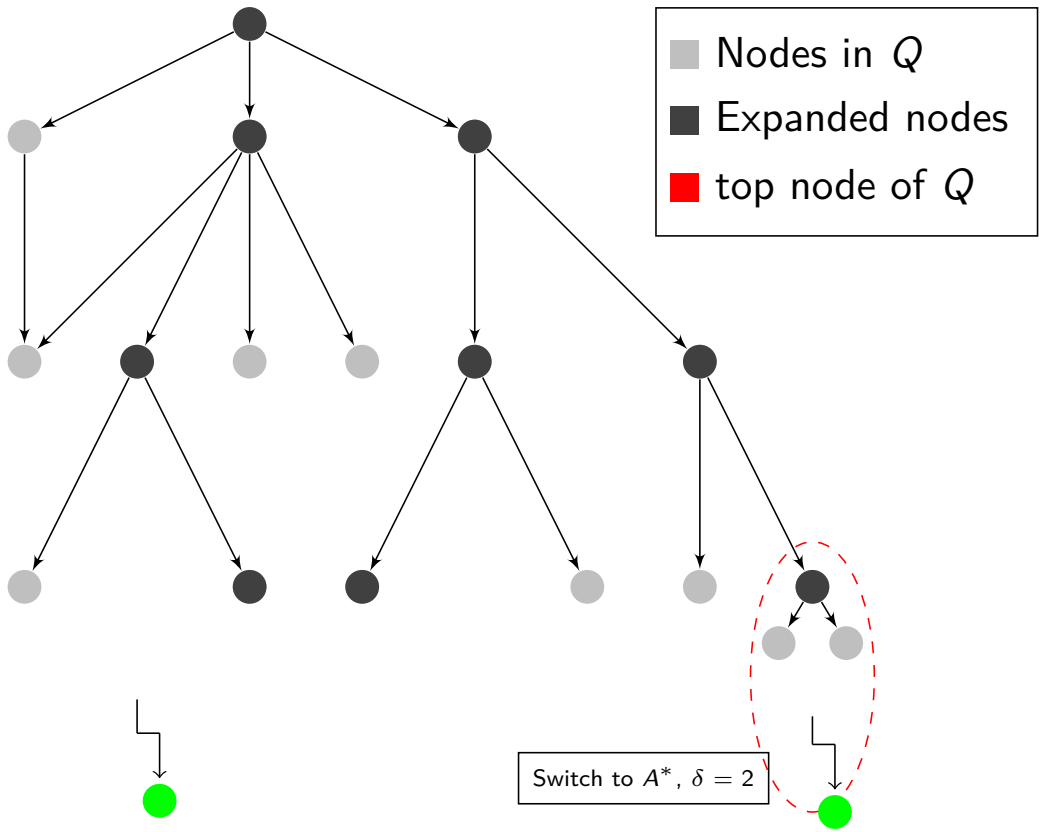
- BS with beam width β: initial beam takes the top node of Q



The A*+BS Hybrid

Heuristic technique we use:

- BS with beam width β: initial beam takes the top node of Q



Bottleneck of the A*+BS concerning LCS solving

1. Only descendant nodes of the root node of each BS visited
2. High beam β required to obtain high-quality solutions:
   - consequently weaker dual bounds obtained

To get rid of these bottlenecks, A*+ACS developed

The A*+ACS Hybrid

Anytime Column Search (Vadlamudi et al. (2012)):

- A major iteration of ACS:
  - at current level of the undergoing state graph (created within prior iterations) expand up to β best not-yet-expanded nodes
  - repeat it level-by-level until bottom has reached
- next iteration: starts with the level closest to root r that has at least one open node

The A*+ACS hybrid:

- BS of the A*+BS replaced by a major ACS iteration
- nodes at the same levels prioritized according to Ex

Experiments

Algorithms involved in our experiments:

1. A*+BS and A*+ACS

2. Anytime Pack Search (APS):
   - our A*+BS where $\delta = 0$, initial beam: top β nodes from Q

3. A*+ACS–DIST: a derivation of A*+ACS where
   - Ex guidance replaced by the heuristic used in Pro–MLCS
   - $\delta = 0$ fixed,

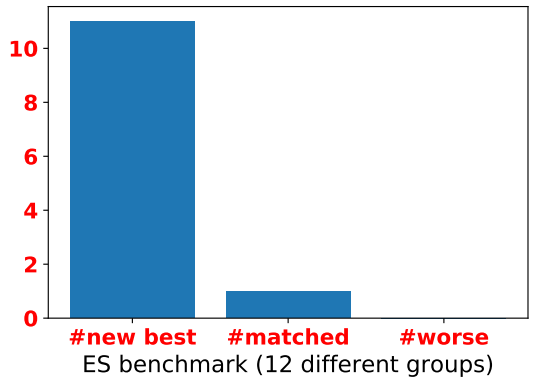
   a reasonable replacement for Pro-MLCS

Irace (Birattari et al. (2010)) used to tune the parameters of our algorithms in both ways:

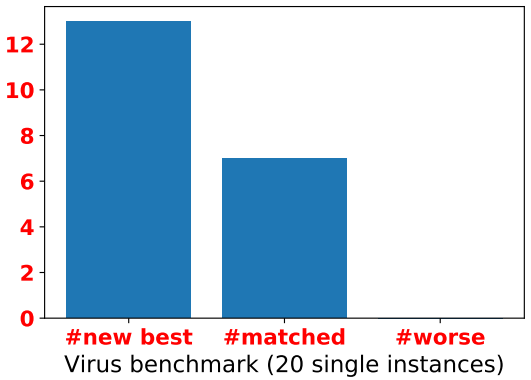
- aiming for high-quality solutions
- aiming for minimizing gaps

Comparison of the Results on Various Benchmark Sets

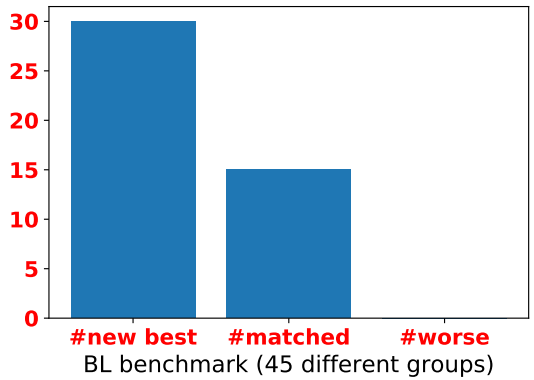
A*+ACS results compared to the (maximum sols of) other anytime algorithms + best results known from literature



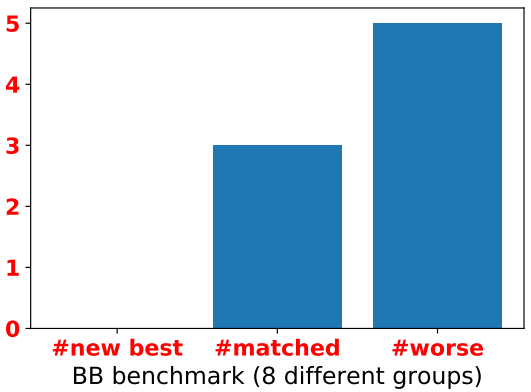
(a) ES benchmark



(b) VIRUS benchmark

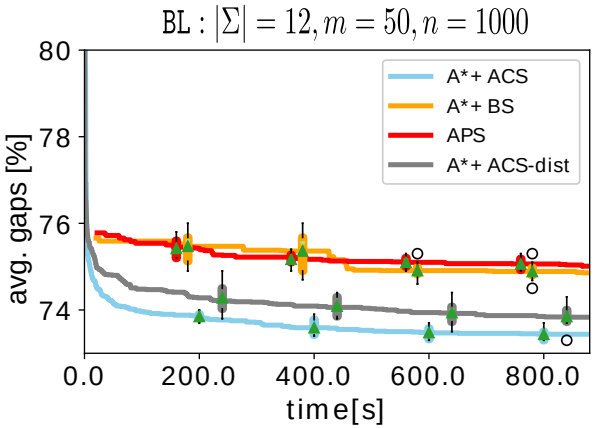


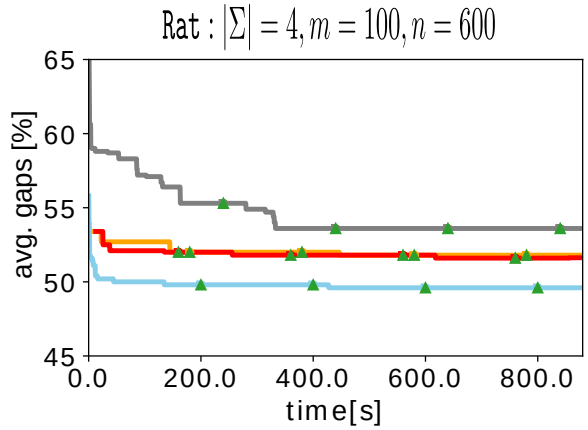
(c) BL benchmark

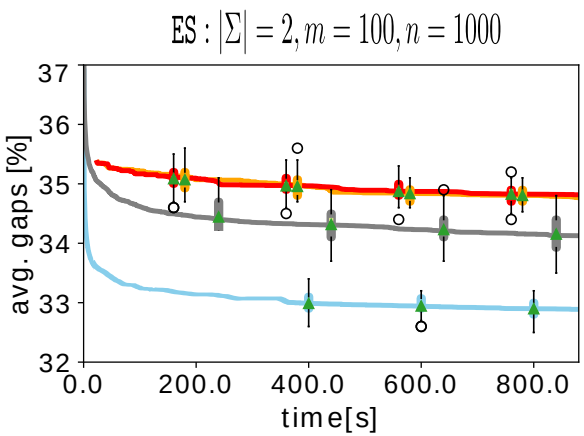


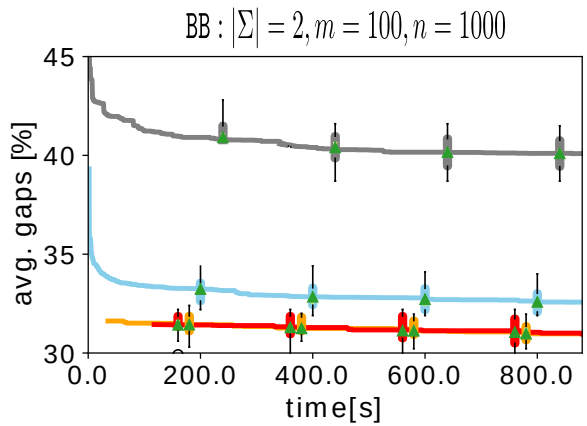
(d) BB benchmark

Instances Tuned for Small Gaps









Conclusions

- Exact approaches to solve the general LCS problem proposed: A*, A*+BS and A*+ACS
- $\mathrm{A}^*+\mathrm{ACS}$ performs best concerning solution quality and small gaps on many instances groups:
  - produces new best solutions (on $\approx 70\%$ instance groups)
  - the difference is significant on many benchmark sets
  - ensures a better balance between primal solutions and gaps quality

C Blum, M Djukanovic, A Santini, H Jiang, CM Li, F Manya, GR Raidl. Solving longest common subsequence problems via a transformation to the maximum clique problem. Computers & Operations Research 125, 105089

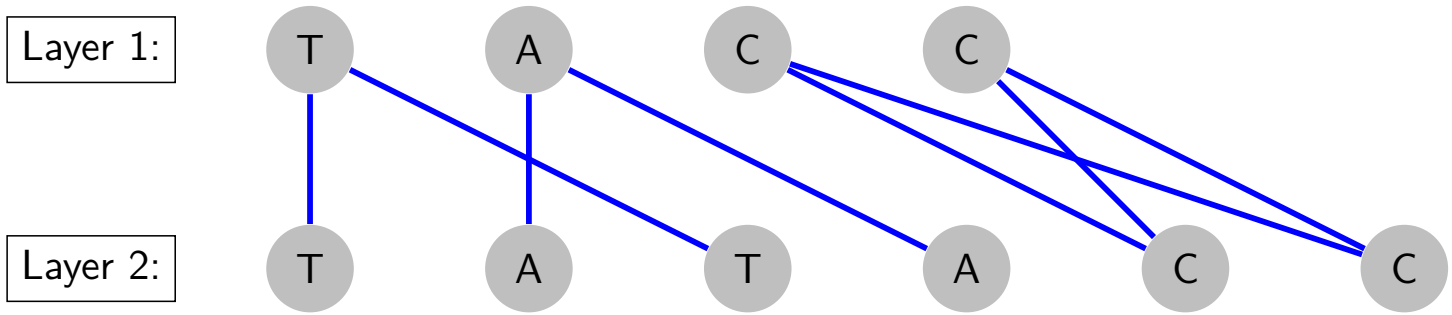
Max-Clique (MC) Problem

- MC problem:
  - Input: Graph $G = (V, E)$
  - Output: Find $V' \subseteq V$ of the maximum cardinalty. s.t. for all $v_i, v_j \in V'$, $v_i v_j \in E$.
  - $\mathcal{NP}$–hard
- Maximum Independent Set (MIS) problem:
  - Find $V' \subseteq V$ of the maximum cardinalty s.t. for all $v_i, v_j \in V'$, $v_i v_j \notin E$ holds
- MC on $G \Longleftrightarrow$ MIS on the corresponding complement graph $\overline{G}$

MC solvers

- A basic ILP model for MIS
- Currently one of best exact solver
  - LMS (2017), suited for large sparse graphs: an incremental MaxSAT reasoning in a B&B scheme
- Currently best-performing heuristic algorithm
  - LSCC-BMS (2016): strong configuration checking to reduce cycling in local search

Transformation to Conflict Graphs ($m = 2$)



- Edges: $p = (i, j) : s_1[i] = s_2[j] = \ell(p)$: matches

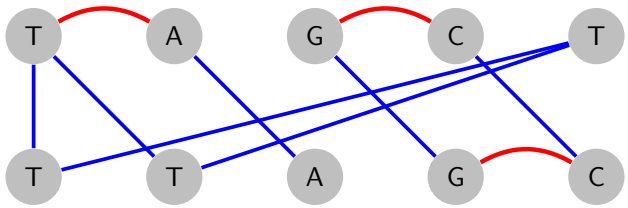
- Edges: (1, 1), (1, 3)
- Edges (1,3) and (2,2) cross (intersection between them)

Conflict Graph for LAPCS Problem

Two edges p and q are in conflict if either

- p and q cross each other.
- p and q do not cross each other, but the arc annotations not preserved

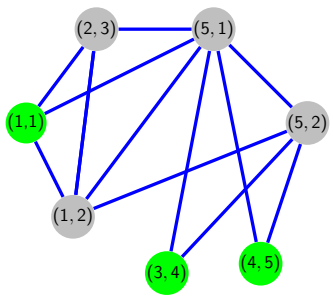
Fig.: Example of an LAPCS instance.



Conflict graph $G^c = (V^c, E^c)$:

- $V^c = \{v_p \mid p \text{ is an edge}\}$
- $E^c = \{v_p v_q \mid p \text{ is in conflict with } q\}$

Fig.: Corresponding conflict graph.

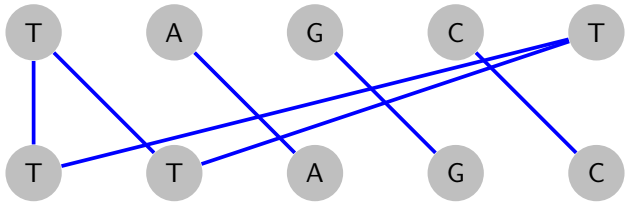


Conflict Graph for RFLCS Problem

Two edges p and q are in conflict if either

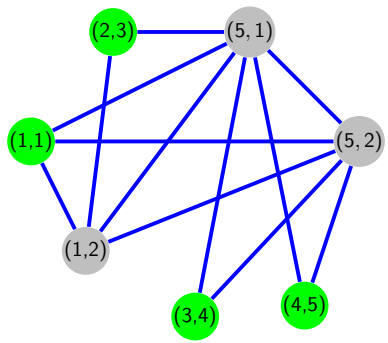
1. p and q cross each other, or
2. p and q cover the same letter: $\ell(p) = \ell(q)$.

Fig.: Example of an RFLCS instance.



Conflict graph $G^c = (V^c, E^c)$: the same as for LCPS problem

Fig.: Corresponding conflict graph.



A Reduction of the Conflict Graphs

- The size of conflict graphs grows quickly in instance size
- Problem-specific information: best available lower bounds, upper bounds utilized

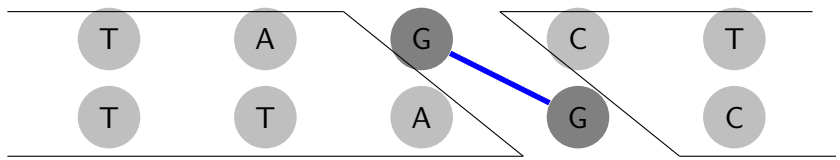
- Every node v_p associated with edge $p = (p_1, p_2)$ separates set S into two (left and right) sub-instances

A Reduction of the Conflict Graphs

- The size of conflict graphs grows quickly in instance size
- Problem-specific information: best available lower bounds, upper bounds utilized

- Every node v_p associated with edge $p = (p_1, p_2)$ separates set S into two (left and right) sub-instances

  Example: S={TAGCT, TTAGC}, $v_p = (3, 4)$



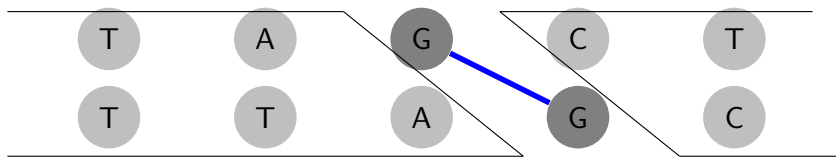
$S_L^p = \{TA, TTA\}, S_R^p = \{CT, C\}$.

A Reduction of the Conflict Graphs

- The size of conflict graphs grows quickly in instance size
- Problem-specific information: best available lower bounds, upper bounds utilized

- Every node v_p associated with edge $p = (p_1, p_2)$ separates set S into two (left and right) sub-instances

  Example: S={TAGCT, TTAGC}, $v_p = (3, 4)$



  $S_L^p = \{TA, TTA\}, S_R^p = \{CT, C\}$.

- If $\mathrm{UB}(S_L^p) + 1 + \mathrm{UB}(S_R^p) < lb$ where
  - UB() is an upper bound of the problem
  - lb: a lower bound on instance $S \Longrightarrow v_p$ safely omitted from $V(G^c)$

Upper and Lower Bounds

- Upper bounds for RFLCS problem:
  - $\mathrm{UB}_1^{\mathrm{RFLCS}}()$: tightening of the occurrence-based upper bound for LCS
  - $\mathrm{UB}_2()$: DP for LCS

Upper and Lower Bounds

- Upper bounds for RFLCS problem:
  - $\mathrm{UB}_1^{\mathrm{RFLCS}}()$: tightening of the occurrence-based upper bound for LCS
  - $\mathrm{UB}_2()$: DP for LCS
- RFLCS problem utilizes $\mathrm{UB}() = \min\{\mathrm{UB}_1^{\mathrm{RFLCS}}(), \mathrm{UB}_2()\}$;
- LAPCS problem utilizes $\mathrm{UB}() = \mathrm{UB}_2()$

lb: outcomes of the state-of-the-art heuristic solvers

Effects of the Conflict Graph Reduction: LAPCS Problem

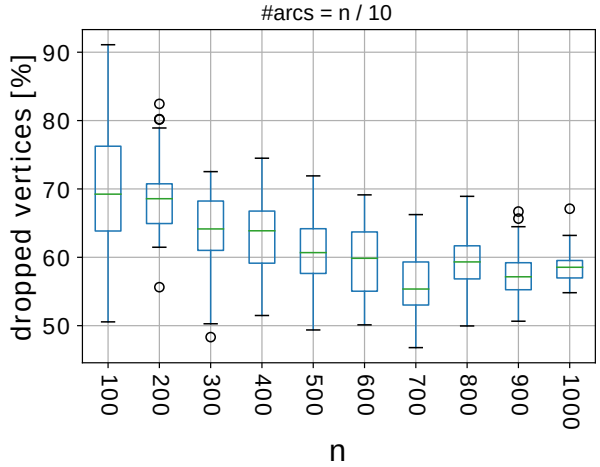


Fig.: Graph reduction (in %) for LAPCS instances from set LAPCS-ARTI.

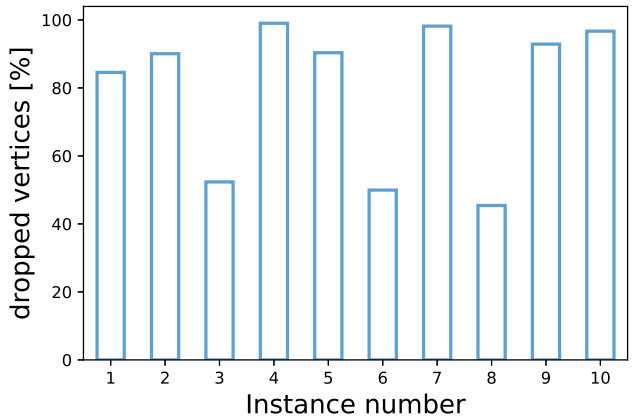


Fig.: Graph reduction (in %) for LAPCS instances from set LAPCS-REAL.

Exact Methods and Reduction

Table: Differences in performance of the exact methods (CPLEX and LMC) summarized for the four different data sets.

Data set	CPLEX		LMC	
	E-M1	**E-M2**	**E-M1**	**E-M2**
RFLCS-1	172.77	257	32.99	84
RFLCS-2	105.19	133	14.90	10
LAPCS-ARTI	526.45	30	20.02	0
LAPCS-REAL	– –	7	– –	5

1. **Measure E-M1**: the average time saving for proving optimality.
2. **Measure E-M2**: indicates the number of instances additionally solved to optimality after graph reduction.

Conclusions

- We proposed a unified approach to solve LCS-related problems (by means of MC solvers)
- We proposed a simple but effective reduction of the size of conflict graphs: high-quality lower and upper bounds
- The reduction beneficial for
  - RFLCS problem:
    - $\approx$ 90% instances solved to optimality
    - CPLEX + reduction outperforms the best exact MC solver in terms of the number of solved instances
  - LAPCS problem:
    - 7 out od 10 real world instances are solved for the first time (CPLEX + reduction)
    - smaller instances solved to optimality (110 out of 900)

General Conclusions & Future Work

- Developed novel heuristic approaches to solve LCS–related problems:
  1. general BS framework: LCS, LCPS, CLCS
  2. novel probability–based and expected length calculation heuristics: LCPS, CLCS, LCS
  3. hybrids of BS and VNS: LCSqS and GCLCS problems
- Exact A* approach for small-to-middle sized instances:
  1. for LCS, LCPS and CLCS problems
  2. new state-of-the art on 2–CLCS, 2–LCPS problems and LCS problem
- Novel anytime hybrids to tackle large–sized instances:
  1. A*+BS and A*+ACS
     - applied on LCS and LCPS
     - A*+ACS state-of-the art for LCS, LCPS
  2. optimality gaps provided for the first time in the literature
- A unified approach based on transforming LCS–related problems to MC problem proposed

Future Work

- Consider another LCS–related problems:
  - the restricted LCS problem, Gotthilf, et al. (2010)
  - the heaviest common subsequence, Jacobson et al. (1992)
  - 2D–LCS problem (strings as matrices), Amir et al. (2008)
- Consider far-from-being-uniform instances for LCS problem
  - input strings that come from more general distributions like multi-nomial distributions (spoken language)
- Consider context of words meaning in solving LCS

Thank you for your attention!

Are there any questions?