

Exact and Heuristic Approaches for the Longest Common Subsequence Problem

PhD seminar

Marko Djukanovic¹, Günther Raidl¹, and Christian Blum²

¹Institute of Logic and Computation, TU Wien, Vienna, Austria,

² Artificial Intelligence Research Institute (IIIA-CSIC),
Campus UAB, Bellaterra, Spain

June 19, 2019



ALGORITHMS AND
COMPLEXITY GROUP



- A *string* is a finite sequence of characters over (finite) alphabet Σ .
- String problems in bioinformatics:
 - ▶ comparing molecules
 - ▶ similarities between molecules: solving combinatorial (optimization) problems
- A *subsequence* of string s is any string obtained from s by deleting zero or more characters.

- *Common subsequences*: measure of similarity
- **LCS problem**:
 - ▶ **Input**: A set of strings $S = \{s_1, \dots, s_m\}$, $m \in \mathbb{N}$, and an alphabet Σ .
 - ▶ **Task**: find a *subsequence* of *maximum* length that is *common* for all strings from S .
 - ▶ **Example**: $|S| = 2$, $S = \{abcbcd a, accbccaa\}$, $\Sigma = \{a, b, c, d\}$
LCS: *abcaa*.

- *Common subsequences*: measure of similarity
- **LCS problem**:
 - ▶ **Input**: A set of strings $S = \{s_1, \dots, s_m\}$, $m \in \mathbb{N}$, and an alphabet Σ .
 - ▶ **Task**: find a *subsequence* of *maximum* length that is *common* for all strings from S .
 - ▶ **Example**: $|S| = 2$, $S = \{\text{abca}bcd\text{a}, \text{acc}bc\text{caa}\}$, $\Sigma = \{a, b, c, d\}$
LCS: *abcaa*.
 - ▶ Solvable in $O(n^m)$ time if m fixed (Dynamic Programming (DP), $n = \max\{|s_i| \mid i = 1, \dots, m\}$)
 - ▶ \mathcal{NP} -hard if m arbitrary

Heuristic approaches from literature for solving LCS:

2001●	Huang et al.: Greedy heuristic (GH) .
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 (MT): a novel BS approach (BS-H) .
2013●	A Hyper-heuristic (HH) approach + new heuristic guidance (HH, BS-Pow).
2019●	LOD 2019: A general BS Framework and a new heuristic guidance (BS-Ex).

Timeline of the exact approaches for solving LCS:

2010●	Wang et al.: A* method .
2013●	Yang et al.: Progressive anytime algorithm (Pro-MLCS) .
2014●	Yang et al.: SA/SLA MLCS anytime algorithms.
2019●	Two A*-based anytime algorithms: A*+BS and A*+ACS (working in progress).

- Many of the approaches from the literature are BS-based (BS is a “limited” BFS)
- Significant differences among them occur (branching, heuristic guidances used)
- Computational studies from the (LCS) literature:
 - ▶ no fully-fair comparisons made
 - ▶ no full-tuning of parameters reported
 - ▶ for some of the approaches mistakes reported (never fixed)
 - ▶ some approaches from literature not tested on each of the LCS benchmarks

- $S[p^L] = S[(p_1^L, \dots, p_m^L)] := \{s_1[p_1^L, |s_1|], \dots, s_m[p_m^L, |s_m|]\}$:
the part of strings from S w.r.t. position vector $p^L \in \mathbb{N}^m$
(subproblems)

General search space for LCS problem: notation

- $S[p^L] = S[(p_1^L, \dots, p_m^L)] := \{s_1[p_1^L, |s_1|], \dots, s_m[p_m^L, |s_m|]\}$:
the part of strings from S w.r.t. position vector $p^L \in \mathbb{N}^m$
(subproblems)
- Example: $S = \{abbbbcbba, accbcbcb, cbbacbcba\}$, and
 $p^L = (2, 5, 3) \Rightarrow$
 $S[(2, 5, 3)] = \{\cancel{a}b\cancel{b}b\cancel{b}cbba, \cancel{a}c\cancel{c}b\cancel{c}bcb, \cancel{c}b\cancel{b}a\cancel{c}bcba\}$

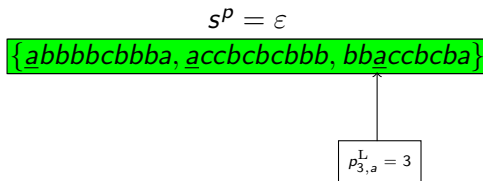
General search space for LCS problem: notation

- $S[p^L] = S[(p_1^L, \dots, p_m^L)] := \{s_1[p_1^L, |s_1|], \dots, s_m[p_m^L, |s_m|]\}$:
the part of strings from S w.r.t. position vector $p^L \in \mathbb{N}^m$
(subproblems)
- Example: $S = \{abbbbcbba, accbcbcb, cbbacbcba\}$, and
 $p^L = (2, 5, 3) \Rightarrow$
 $S[(2, 5, 3)] = \{\cancel{a}b\cancel{b}b\cancel{b}cbba, \cancel{a}c\cancel{c}b\cancel{c}bcb, \cancel{c}b\cancel{b}a\cancel{c}bcba\}$
- String s^p is a *partial solution* of S iff it is a common subs. of all strings from S

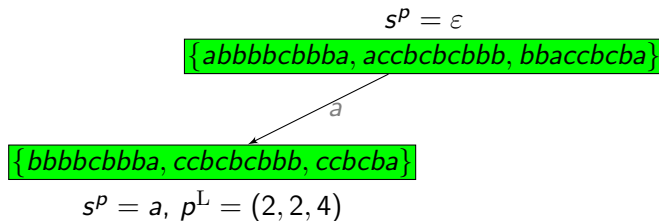
$$s^P = \varepsilon$$

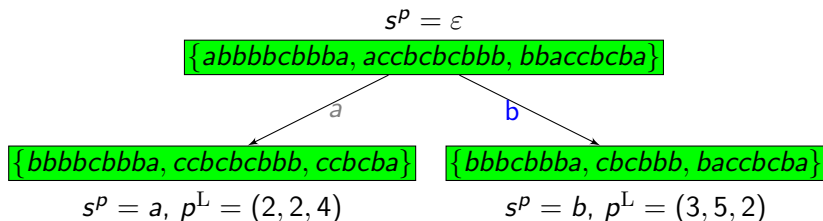
$\{abbbbcbbba, accbcbcbbb, bbaccbcba\}$

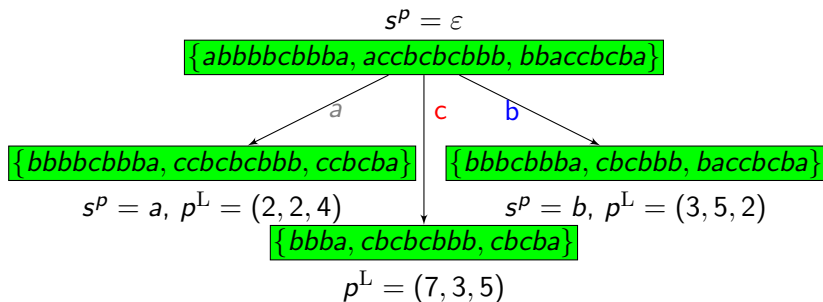
The general search framework

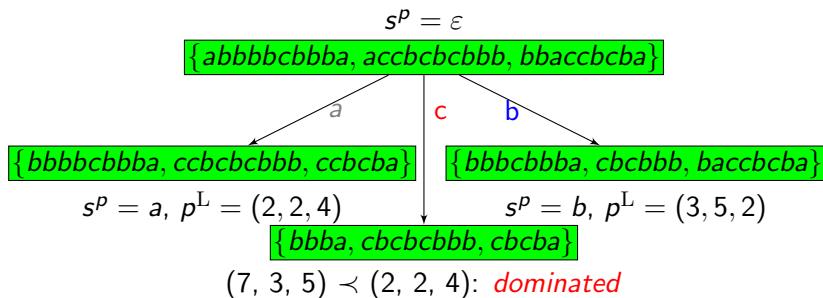


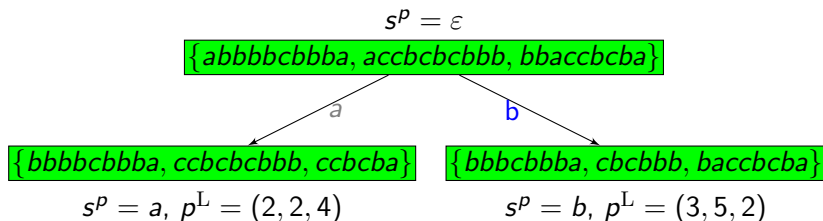
The general search framework











- Node $v := (p^{L,v}, l^v)$:
 - ▶ $S[p^{L,v}]$: remaining strings relevant to extend partial solution
 - ▶ l^v : length of the corresp. partial solution
- Expansions of v :
 - ▶ determine non-dominated feasible letters for $S[p^{L,v}]$
 - ▶ extend its partial solution in all possible ways: determine $p^{L,v'}$ (and $l^{v'} = l^v + 1$) of each child v' , accordingly
 - ▶ complete nodes = non-expanded nodes
- Evaluation of v :
 - ▶ upper bound functions
 - ▶ heuristics which do not provide bounds
- $p^{L,r} = (1, \dots, 1)$: the root position vector (corresp. to the whole problem S) and $r = (p^{L,r}, 0)$ is the root node

Algorithm 1 The General BS Framework (GBSF) for LCS.

- 1: **Input:** an instance (S, Σ) ; h : heuristic to evaluate nodes;
 ub_{prune} : heuristic to prune nodes;
 k_{best} : filtering of the expanded nodes (dominance relation check);
 β : beam size... and possibly some other params
- 2: **Output:** a feasible LCS solution
- 3: $B \leftarrow \{r\}$
- 4: $s_{\text{lcs}} \leftarrow \varepsilon$
- 5: **while** $B \neq \emptyset$ **do**
- 6: $V_{\text{ext}} \leftarrow \text{ExtendAndEvaluate}(B, h, n_{\text{best}} = |V_{\text{ext}}|)$
- 7: update s_{lcs} if a complete node v with a new largest l_v value reached
- 8: $V_{\text{ext}} \leftarrow \text{Prune}(V_{\text{ext}}, ub_{\text{prune}})$ // optional
- 9: $V_{\text{ext}} \leftarrow \text{Filter}(V_{\text{ext}}, k_{\text{best}})$ // optional
- 10: $B \leftarrow \text{Reduce}(V_{\text{ext}}, \beta)$
- 11: **end while**
- 12: return s_{lcs}

- **ExtendAndEvaluate** and **Reduce**: basic procedures of each BS
- n_{best} : reduce V_{ext} to an amount of the n_{best} best nodes
- **Prune**($V_{\text{ext}}, ub_{\text{prune}}$):
 - ▶ ub_{prune} : a real upper bound function
 - ▶ For example: node v where $l^v + ub_{\text{prune}}(v) \leq |s_{\text{lcs}}|$ dropped out from V_{ext}
- **Filter**($V_{\text{ext}}, k_{\text{best}}$):
 - ▶ take the $k_{\text{best}} \leq |V_{\text{ext}}|$ nodes which are the highest in priority: P_{reduce}
 - ▶ $(\forall u \in V_{\text{ext}})$, if $\exists u' \in P_{\text{reduce}}$ s.t. $u \prec u'$: u dropped out from V_{ext}
 - ▶ $k_{\text{best}} = |V_{\text{ext}}|$: full *dominate-filtering*, otherwise the *reduced-filtering* executed

Table: The upper bound functions from the literature.

1995	•	$\widetilde{\text{UB}}_1(v) = \min_{i=1,\dots,m} s_i - p_i^{L,v} + 1 .$
2009	•	$\text{UB}_1(v) = \sum_{a \in \Sigma} \min_{i=1,\dots,m} s_i[p_i^{L,v}, s_i] _a.$
2010	•	$\text{UB}_2(v)$: DP-based.

Table: The upper bound functions from the literature.

1995	•	$\widetilde{UB}_1(v) = \min_{i=1,\dots,m} s_i - p_i^{L,v} + 1 .$
2009	•	$UB_1(v) = \sum_{a \in \Sigma} \min_{i=1,\dots,m} s_i[p_i^{L,v}, s_i] _a.$
2010	•	$UB_2(v)$: DP-based.

$\Rightarrow \text{UB}_{\min} = \min\{UB_1, UB_2\}$ at the moment the tightest known.

Table: The timeline of the heuristics developed for LCS.

2001●	Greedy heuristic: $g(v, a)$, $a \in \Sigma$ ("skip letters" measure).
2009●	Rank-based heuristic: $\text{Rank}(v)$.
2012●	Prob.-based heuristic: $H(v)$.
2013●	Power heuristic: $\text{Pow}(v)$.
2019●	Expected length heuristic: $\text{EX}(v)$.

Table: The timeline of the heuristics developed for LCS.

2001●	Greedy heuristic: $g(v, a)$, $a \in \Sigma$ ("skip letters" measure).
2009●	Rank-based heuristic: $\text{Rank}(v)$.
2012●	Prob.-based heuristic: $H(v)$.
2013●	Power heuristic: $\text{Pow}(v)$.
2019●	Expected length heuristic: $\text{EX}(v)$.

⇒ state-of-the-art heuristic before Year 2019: **H(v)** and **Pow(v)**

Blum et al. (2009):

- the expansions on the same level ranked w.r.t. \widetilde{UB}_1 or $g(\cdot, \cdot)$:
 - ▶ the node which g (or \widetilde{UB}_1) value is the largest one in V_{ext} receives **rank** 1, the second largest receives rank 2...
- **Rank**(\cdot)-value of any node cumulatively calculated by:
 - ▶ $\text{Rank}(v) = \text{Rank}(\text{Parent}(v)) + \text{rank}(v)$, $v \in V_{\text{ext}}$, where $\text{Rank}(r) = 1$ and $\text{rank}(v)$ is the rank of v received from the corresponding level

- MT, 2012: $P(p, q)$ = probability that any string s of length p is a subsequence of a random string of length q (matrix P created by DP)
- $\Rightarrow \text{Prob}(s \prec_{\text{subs}} S[p^{\text{L},v}]) = \prod_{i=1}^m P(p, |s_i| - p^{\text{L},v} + 1) := H(v, p)$
- p heuristically determined at each level of BS by:

$$p = \min \left\{ \frac{\min_{v \in V_{\text{ext}}} \widetilde{\text{UB}}_1(v)}{|\Sigma|}, 1 \right\}$$

- A generalization of \widetilde{UB}_1
- $\text{Pow}(v) = \left(\prod_{i=1}^m (|s_i| - p_i^{L,v} + 1) \right)^{\rho(\cdot)} \times \widetilde{UB}_1(v), \rho \in [0, 1)$
where $\rho(\cdot)$ carefully chosen s.t. $\lim_{m \rightarrow \infty} \rho(m, \cdot) \rightarrow C \approx 0$.
- MT, 2013:

$$\rho(m, a, b, c) = a \cdot \exp(-b \cdot m) + c, \quad a, b, c \geq 0.$$

- State-of-the-art approaches from the literature expressible in terms of the GBSF
- Able to ensure fully-fair comparisons among the known approaches
- All ideas “in one place”: all the approaches united into one

- Setting:

- ▶ $h = Rank^{-1}$, $ub_{prune} = UB_1$, $n_{best} = \lfloor \beta \cdot \mu \rfloor$ and no Filter used \Rightarrow BS-BLUM covered
- ▶ $h = ub_{prune} = UB_2$ with no Filter and additional Prune: node $v \in V_{ext}$ whereas $\max_{v' \in V_{ext}} ub_{prune}(v') - ub_{prune}(v) > W \Rightarrow$ BS-WANG covered
- ▶ $h = H$ and execute Filter procedure \Rightarrow BS-H covered
- ▶ $h = Pow$ and execute Filter procedure \Rightarrow BS-POW covered

- Setting:
 - ▶ $h = Rank^{-1}$, $ub_{prune} = UB_1$, $n_{best} = \lfloor \beta \cdot \mu \rfloor$ and no Filter used \Rightarrow BS-BLUM covered
 - ▶ $h = ub_{prune} = UB_2$ with no Filter and additional Prune: node $v \in V_{ext}$ whereas $\max_{v' \in V_{ext}} ub_{prune}(v') - ub_{prune}(v) > W \Rightarrow$ BS-WANG covered
 - ▶ $h = H$ and execute Filter procedure \Rightarrow BS-H covered
 - ▶ $h = Pow$ and execute Filter procedure \Rightarrow BS-POW covered
- We emphasize: making use of $ub_{prune} := UB_{min}$ in BS-BLUM and BS-WANG \Rightarrow better results
- HH approach: BS-H and BS-POW executed with a small beam size β_{low} to check if Pow or H is a better guidance (based on the solutions obtained); execute a BS with the winning heuristic as its guidance where $\beta \gg \beta_{low}$

A novel heuristic for the LCS problem

- General assumption:
 - ▶ input strings are uniform random strings
 - ▶ no independence among strings occurred
- MT, 2012:
 - ▶ $P(p, q)$: probability that any string of length p is a subsequence of random string of length q expressed by a matrix P
 - ▶ $\Rightarrow \text{Prob}(s \prec_{\text{subs}} S) = \prod_{i=1}^m P(|s|, |s_i|)$

A novel heuristic for the LCS problem

- General assumption:

- ▶ input strings are uniform random strings
- ▶ no independence among strings occurred

- MT, 2012:

- ▶ $P(p, q)$: probability that any string of length p is a subsequence of random string of length q expressed by a matrix P
- ▶ $\Rightarrow \text{Prob}(s \prec_{\text{subs}} S) = \prod_{i=1}^m P(|s|, |s_i|)$

- Construction

- ▶ Sb_k : all subsequence of length k over Σ
- ▶ $x \in \text{Sb}_k \mapsto E_{v_x}$: the event that x is a common subsequence of every string in S
- ▶ **Assumption**: for $x, y \in \text{Sb}_k$, E_{v_x} is independent of E_{v_y}

- Moreover:

- ▶ Y : a binary random variable denoting the length of an LCS
 - ▶ $Y_k \in \{0, 1\}$ a random variable indicating if the strings from S have an LCS of length $k \geq 0$ or not
- $\Rightarrow \mathbb{E}[Y_k] - \mathbb{E}[Y_{k+1}]$: prob. that the length of an LCS is exactly k

- $\mathbb{E}[Y] = \sum_{k=1}^{\widetilde{UB}_1(r)} k \cdot \text{Prob}(Y = k) = \sum_{k=1}^{\widetilde{UB}_1(r)} \mathbb{E}[Y_k]$

- $1 - \mathbb{E}[Y_k] = (1 - \prod_{i=1}^m P(k, |s_i|))^{|\Sigma|^k} \Rightarrow$

$$\text{EX}(v) = \text{EX}(p^{L,v}) = \sum_{k=1}^{\widetilde{UB}_1(v)} 1 - \left(1 - \prod_{i=1}^m P(k, |s_i| - p_i^{L,v} + 1) \right)^{|\Sigma|^k} \quad (1)$$

- Numerical stability: $|\Sigma|^k$ can be extremely large \Rightarrow **power-decomposition** utilized (use a power $k' \ll k$ multiple times)
- Runtime of EX: improved from $O(m \cdot n)$ to the expected runtime $O(m \cdot \log n)$ by applying **divide-and-conquer**
- Setting $h = \text{EX}$, no pruning, use `Filter()` \Rightarrow **BS-Ex**

Settings:

- GBSF implemented in C++
- single-threaded mode on an Intel Xeon E5-2640 with 2.40GHz
- 16 GB of memory

Benchmarks:

- Rat, Virus, Random (Shyu and Tsai , 2009): 20×3 the first two groups **biologically** inspired
- ES benchmark (Easton and Singireddy, 2008): 600 instances
- BB benchmark (Blum and Blesa, 2007): 80 instances – **increased similarity/dependence** between input strings
- BL benchmark (Blum, 2016): 450 instances

Parameters:

- results largely depend on the size of β :
 - ▶ *low-time* setting: $\beta = 50$
 - ▶ *high-quality* setting: $\beta = 600$
- **BB** benchmark differs from the others because of the **similarity** between input strings: the algorithms tuned **separately**

¹Our re-implementation of the five competitor algorithms from the literature reported equally good (or even better instances) on all benchmarks that those reported in the papers.

Parameters:

- results largely depend on the size of β :
 - ▶ *low-time* setting: $\beta = 50$
 - ▶ *high-quality* setting: $\beta = 600$
- BB benchmark differs from the others because of the similarity between input strings: the algorithms tuned separately
- Irace report (no independence between input strings):¹
 - ▶ BS-Blum: UB_{\min} , $g(\cdot, \cdot)$, $\mu = 4.0$ and $k_{\text{best}} = 5$
 - ▶ BS-WANG: $W = 10$
 - ▶ BS-POW: $k_{\text{best}} = 100$, $a = 1.677$, $b = 0.054$, and $c = 0.074$
 - ▶ BS-H: $k_{\text{best}} = 50$
 - ▶ BS-Ex: $k_{\text{best}} = 100$
 - ▶ HH: $\beta_{\text{low}} = 50$

¹Our re-implementation of the five competitor algorithms from the literature reported equally good (or even better instances) on all benchmarks that those reported in the papers.

lrace reported for BB benchmark:

- BS-BLUM: UB_{\min} , $\mu = 4.0$ and $k_{\text{best}} = 1000$
- BS-WANG: $W = 10$
- BS-POW: $k_{\text{best}} = 100$, $a = 1.823$, $b = 0.112$, and $c = 0.014$
- BS-H: $k_{\text{best}} = 100$
- **BS-EX: $k_{\text{best}} = 100$**
- HH: $\beta_{\text{low}} = 50$

Table: Results on benchmark set Rat.

Σ	n	m	<i>low-time, literature</i>			<i>low-time, Bs-Ex</i>		<i>high-quality, literature</i>			<i>high-quality, Bs-Ex</i>	
			\bar{s}_{best}	\bar{t}_{best}	Algo.	\bar{s}_{best}	$\bar{t}[s]$	\bar{s}_{best}	\bar{t}_{best}	Algo.	\bar{s}_{best}	$\bar{t}[s]$
4	600	10	201	0.09	Bs-Pow	198	0.22	204	1.18	Bs-Pow	*205	3.09
4	600	15	182	0.10	Bs-Pow	182	0.18	184	0.62	Bs-H	*185	2.65
4	600	20	169	0.05	Bs-Pow	168	0.15	170	0.94	Bs-Pow	*172	2.25
4	600	25	166	0.12	Bs-Pow	167	0.18	168	1.01	Bs-Pow	*170	2.71
4	600	40	151	0.04	Bs-H	146	0.15	150	1.02	Bs-Pow	152	1.81
4	600	60	149	0.10	Bs-Pow	150	0.17	151	1.16	Bs-Pow	*152	2.27
4	600	80	137	0.05	Bs-H	137	0.17	139	0.67	Bs-H	*142	2.47
4	600	100	133	0.07	Bs-Pow	131	0.14	135	0.47	Bs-H	*137	2.50
4	600	150	125	0.06	Bs-H	127	0.13	126	0.91	Bs-Pow	*129	1.97
4	600	200	121	0.09	Bs-Pow	121	0.17	*123	0.70	Bs-Pow	*123	2.65
20	600	10	70	0.09	Bs-H	70	0.37	*71	1.86	Bs-H	*71	3.44
20	600	15	61	0.15	Bs-Pow	62	0.28	62	1.40	Bs-H	*63	2.55
20	600	20	53	0.12	Bs-Pow	53	0.20	54	1.15	Bs-H	54	2.45
20	600	25	50	0.22	Bs-WANG	50	0.21	51	1.09	Bs-H	*52	2.94
20	600	40	48	0.09	Bs-H	47	0.19	49	1.15	Bs-BLUM	49	2.97
20	600	60	46	0.09	Bs-H	46	0.20	47	1.61	Bs-Pow	46	2.42
20	600	80	43	0.18	Bs-BLUM	41	0.21	*44	1.14	Bs-H	43	2.64
20	600	100	38	0.11	Bs-Pow	38	0.23	39	0.96	Bs-H	*40	2.54
20	600	150	36	0.32	Bs-BLUM	36	0.14	37	5.11	Bs-WANG	37	2.03
20	600	200	34	0.10	Bs-Pow	34	0.18	34	2.62	Bs-BLUM	34	2.74

Table: Results on benchmark BL (averaged, 10 instances per row, $|\Sigma| = 4$).

$ \Sigma $	n	m	<i>low-time, literature</i>			<i>low-time, Bs-Ex</i>		<i>high-quality, literature</i>			<i>high-quality, Bs-Ex</i>	
			$ \bar{s}_{\text{best}} $	\bar{t}_{best}	Algo.	$ \bar{s}_{\text{best}} $	$\bar{t}[s]$	$ \bar{s}_{\text{best}} $	\bar{t}_{best}	Algo.	$ \bar{s}_{\text{best}} $	$\bar{t}[s]$
4	100	10	34.0	0.01	Bs-POW	34.0	0.02	*34.1	0.14	Bs-POW	*34.1	0.39
4	100	50	23.7	0.03	HH	23.8	0.02	24.1	0.08	Bs-H	*24.2	0.30
4	100	100	21.5	0.03	HH	21.5	0.02	*22.0	0.23	Bs-WANG	*22.0	0.27
4	100	150	20.2	0.03	HH	20.2	0.02	*20.5	0.12	Bs-POW	*20.5	0.31
4	100	200	19.8	0.01	Bs-H	19.5	0.02	*19.9	0.14	Bs-H	*19.9	0.31
4	500	10	182.0	0.24	HH	181.2	0.25	*184.1	1.03	Bs-POW	184.0	2.41
4	500	50	138.6	0.21	HH	139.1	0.19	140.1	0.90	Bs-POW	*141.0	2.13
4	500	100	129.2	0.06	Bs-H	129.7	0.18	130.2	1.01	Bs-POW	*130.8	2.10
4	500	150	124.7	0.07	Bs-H	125.5	0.19	125.9	0.79	Bs-H	*126.4	2.38
4	500	200	122.6	0.07	Bs-H	123.0	0.22	123.2	0.83	Bs-H	*123.7	2.61
4	1000	10	368.3	0.35	HH	368.5	0.42	373.2	1.80	Bs-POW	*374.6	5.22
4	1000	50	284.2	0.36	HH	286.2	0.35	287.0	1.69	Bs-POW	*288.6	4.43
4	1000	100	267.5	0.11	Bs-H	268.8	0.41	269.5	1.36	Bs-H	*270.6	4.56
4	1000	150	259.5	0.14	Bs-H	261.2	0.47	261.5	1.38	Bs-H	*262.8	5.30
4	1000	200	254.9	0.17	Bs-H	256.0	0.52	256.5	1.81	Bs-H	*257.6	6.31

Table: Results on benchmark set BB (averaged over 10 instances per row).

$ \Sigma $	n	m	<i>low-time, literature</i>			<i>low-time, BS-Ex</i>		<i>high-quality, literature</i>			<i>high-quality, BS-Ex</i>	
			$ \bar{\mathbf{s}}_{\text{best}} $	\bar{t}_{best}	Algo.	$ \bar{\mathbf{s}}_{\text{best}} $	$\bar{t}[s]$	$ \bar{\mathbf{s}}_{\text{best}} $	\bar{t}_{best}	Algo.	$ \bar{\mathbf{s}}_{\text{best}} $	$\bar{t}[s]$
2	1000	10	662.9	0.33	HH	635.1	0.44	*676.5	1.16	BS-H	673.5	5.49
2	1000	100	551.0	0.54	HH	525.1	0.50	*560.7	2.10	BS-POW	536.6	6.05
4	1000	10	537.8	0.43	HH	453.0	0.48	*545.4	1.73	BS-H	545.2	6.24
4	1000	100	371.2	0.24	BS-POW	318.6	0.53	*388.8	2.86	BS-POW	329.5	5.85
8	1000	10	462.6	0.27	BS-BLUM	338.8	0.53	*462.7	7.93	BS-BLUM	*462.7	7.90
8	1000	100	260.9	0.87	BS-BLUM	198.0	0.67	*272.1	18.43	BS-BLUM	210.6	8.00
24	1000	10	385.6	0.67	BS-BLUM	385.6	1.04	385.6	13.14	BS-BLUM	385.6	16.24
24	1000	100	147.0	0.66	BS-POW	95.8	0.98	*149.5	8.01	BS-POW	113.3	12.45

- We presented a general search framework for the LCS problem
- We derived a novel heuristic which approximate the expected length of an LCS:
 - ▶ a new state-of-the-art guidance on all except for BB benchmark
 - ▶ able to produce **new best-known** results in 48 out of 67 cases
- rigorous comparison among the approaches reported

- We presented a general search framework for the LCS problem
- We derived a novel heuristic which approximate the expected length of an LCS:
 - ▶ a new state-of-the-art guidance on all except for BB benchmark
 - ▶ able to produce **new best-known** results in 48 out of 67 cases
- rigorous comparison among the approaches reported

Next steps to do:

- What about exact way of solving the LCS problem? The limitations...

- We presented a general search framework for the LCS problem
- We derived a novel heuristic which approximate the expected length of an LCS:
 - ▶ a new state-of-the-art guidance on all except for BB benchmark
 - ▶ able to produce **new best-known** results in 48 out of 67 cases
- rigorous comparison among the approaches reported

Next steps to do:

- What about exact way of solving the LCS problem? The limitations...⇒ Let's visit the next slides...

Exact Approaches for Solving LCS

- General search space allows us defining exact approaches like A*:
 - ▶ Hart et al. (1968)
 - ▶ best-first-search; widely used in pathfindings/planning
 - ▶ **informed search**: nodes prioritized acc. to $f(v) = g(v) + h(v)$:
 - ★ $g(v)$: the cost from root node to v
 - ★ $h(v)$: a heuristic function, an estimated cost from v to a **goal node** (dual bound)
 - ▶ a priority queue Q maintained: keep not yet expanded (open) nodes

- General search space allows us defining exact approaches like A*:
 - ▶ Hart et al. (1968)
 - ▶ best-first-search; widely used in pathfindings/planning
 - ▶ **informed search**: nodes prioritized acc. to $f(v) = g(v) + h(v)$:
 - ★ $g(v)$: the cost from root node to v
 - ★ $h(v)$: a heuristic function, an estimated cost from v to a **goal node** (dual bound)
 - ▶ a priority queue Q maintained: keep not yet expanded (open) nodes
- **A* search for the LCS problem**
 - ▶ $v = (p^{L,v}, l^v)$, root node $r = ((1, \dots, 1), 0)$
 - ▶ $g(v) = l^v$ and $h(v) = \text{UB}_{\min}(v)$
 - ▶ goal nodes = complete nodes

- A* operates on a **directed acyclic** graph $G(N, A)$:
 - ▶ $v = (p^{L,v}, l^v) \in N$: set of nodes
 - ▶ $a = (v_1, x, v_2) \in A, x \in \Sigma$: set of arcs where
 - ★ v_2 is a child node of v_1
 - ★ partial sol. of v_2 obtained by appending letter x to the part. sol. of v_1 indicates the subproblem $S[p^{L,v_2}]$
 - ▶ top nodes of Q always expanded first
 - ▶ N implemented as a **hash-map**: efficient look-up operations of nodes
 - ▶ update l^v (and priority) of any (created) node when a larger path to the node is encountered
- UB_{\min} is **monotonic**: no re-expansion of already created nodes

A working example of A*...

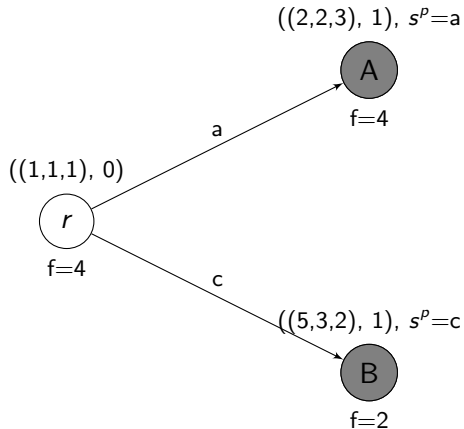
$((1,1,1), 0)$



$f=4$

Instance set: $S = \{abbca, acbca, cacbaa\}$

A working example of A^* ...

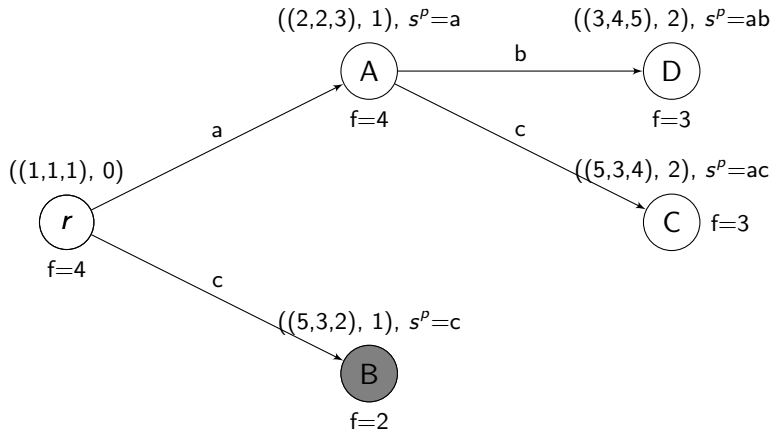


open nodes

complete nodes

Instance set: $S = \{abbca, acbca, cacbaa\}$

A working example of A*...

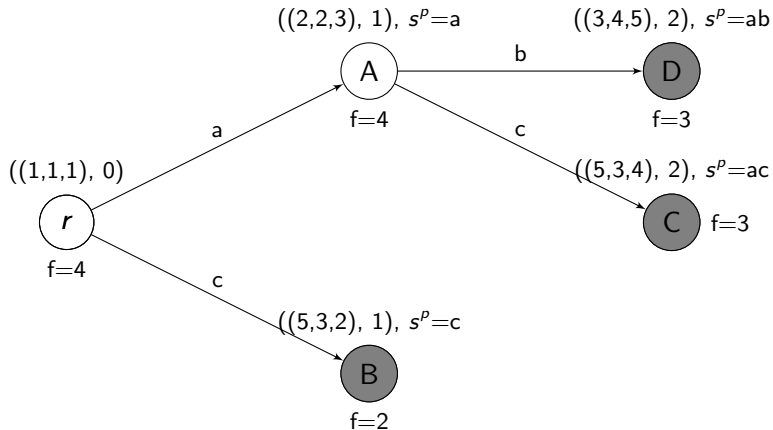


open nodes

complete nodes

Instance set: $S = \{abbca, acbca, cacbaa\}$

A working example of A*...

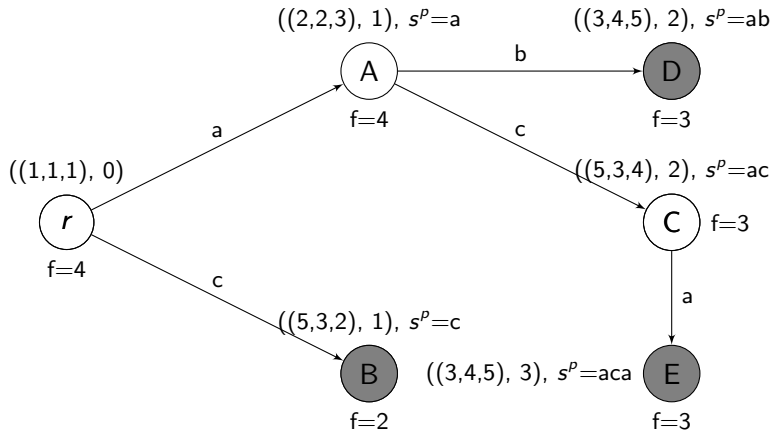


open nodes

complete nodes

Instance set: $S = \{abbca, acbca, cacbaa\}$

A working example of A*...

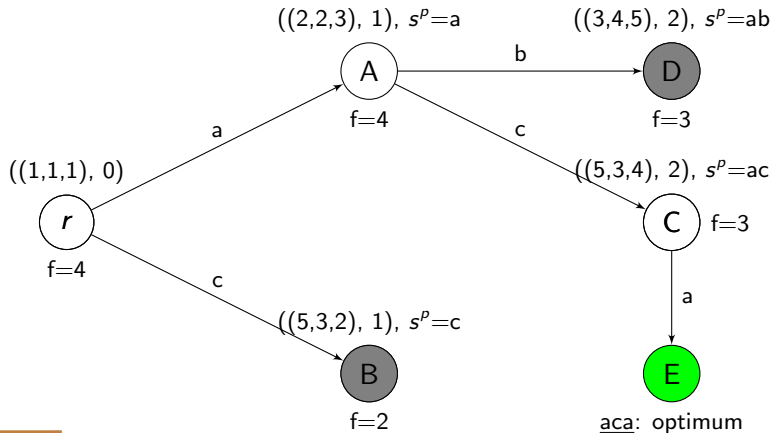


open nodes

complete nodes

Instance set: $S = \{abbca, acbca, cacbaa\}$

A working example of A*...



open nodes

complete nodes

Instance set: $S = \{abbca, acbca, cacbaa\}$

A*: results

- Time limit: 900s; memory limit: 32GB
- A* search: applied on the instances up to $n = 100$.

Table: A* results on the BL benchmark.

m	n	$ \Sigma $	#runs	\bar{s}	std	$\bar{t}[s]$	#opt
10	100	4	10	20.5	17.65	428.33	6
10	100	12	10	12.7	0.82	1.73	10
10	100	20	10	7.9	0.74	0.08	10
50	100	4	10	0.0	0.00	–	0
50	100	12	10	6.9	0.32	0.17	10
50	100	20	10	3.0	0.00	0.06	10
100	100	4	10	0.0	0.00	–	10
100	100	12	10	5.2	0.42	0.08	10
100	100	20	10	2.1	0.32	0.07	10
150	100	4	10	0.0	0.00	–	0
150	100	12	10	4.7	0.48	0.07	10
150	100	20	10	1.9	0.32	0.08	10
200	100	4	10	0.0	0.00	–	0
200	100	12	10	4.1	0.32	0.07	10
200	100	20	10	1.1	0.32	0.06	10

- The phrase “anytime algorithms” from early 80’s
- A class of algorithms which:
 - ▶ return solutions of reasonable quality when terminated
 - ▶ improve solutions over time
 - ▶ if enough resources is insured, able to prove optimality
- In the literature:
 - ▶ Likhachev et al. (2004): ARA*, Aine et al. (2007): AWA*, Berg et al. (2011): ANA*, etc.
 - ▶ Zhou and Hansen (2005): Beam stack search (BS-based anytime algorithm)
 - ▶ [Anytime Column Search](#) (2012)
 - ▶ [Anytime Pack Search](#) (2016): state-of-the-art algorithm on various domains: successively apply BS such that (up to) the best *pack* > 0 nodes from Q packed for the initial beam

- General idea: A^* framework used to construct anytime algorithms
- Efficient heuristic search approaches incorporated into A^*
- $A^* + BS$: “embedding” BS into A^* framework
 - 1 a standalone BS with a beam width β performed on the top node of Q
 - 2 when step 1 finished, execute $\delta \geq 0$ iterations of A^* and go to 1
- Limitation of $A^* + BS$:
 - ▶ only descendant nodes of the starting node (of an BS) are expanded

ACS algorithm (Vadlamudi et al., 2012):

- at each level i of the state graph, a priority queue Q_i is maintained
- iteratively expand up to the best β open nodes of each level in a **level-by-level** manner starting from the first non-empty Q_i towards the bottom layer

A* + ACS:

- “Embedded” BS of the A*+BS replaced by a single ACS iteration
- Q_i prioritized according to $h = EX$

- Based on an iterative beam widening search strategy
- Nodes into different layers based on their distance from root r
- p. queues maintained at each level i : $\text{New}_i, \text{Open}_i, \text{Closed}_i$
- $\text{dist}(v) = \sum_{i=1}^m p_i^{L,v}$: prioritize the nodes in New_i and Closed_i , while
 $\widetilde{\text{dist}}(v) = \min_{(v' \in \text{Children}(v) \wedge v' \text{ not exp.})} \text{dist}(v')$: prioritize Open_i

- dist: not a strong heuristic guidance on many occasions
- optimality proven only by processing all nodes
 - ▶ \Rightarrow no proven gaps returned over time
- dominance relation: checked by a complicated multi-dimensional index tree data structure

Anytime comparisons: solution quality

- $A^* + \text{BS}$: $\beta = 100, \delta = 500$
- $A^* + \text{ACS}$; $\beta = 100, \delta = 500, h = \text{EX}$
- APS: $\text{pack} = 100$
- $A^* + \text{ACS-dist}$: $\beta = 100, \delta = 500, h = \text{dist}$

Anytime comparisons: solution quality

- $A^* + BS$: $\beta = 100, \delta = 500$
- $A^* + ACS$; $\beta = 100, \delta = 500, h = EX$
- APS: $pack = 100$
- $A^* + ACS\text{-}dist$: $\beta = 100, \delta = 500, h = dist$

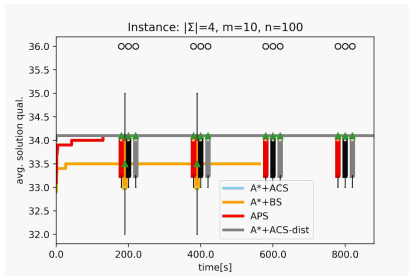


Figure: $m = 10, n = 100, |\Sigma|=4$.

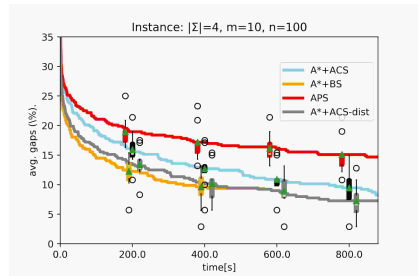


Figure: $m = 50, n = 1000, |\Sigma| = 12$.

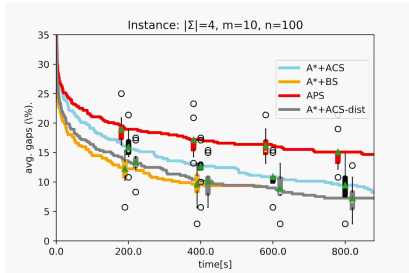


Figure: $m = 10, n = 100, |\Sigma|=4$.

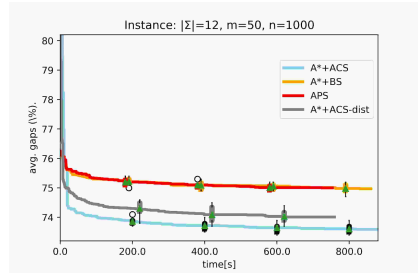


Figure: $m = 50, n = 1000, |\Sigma| = 12$.

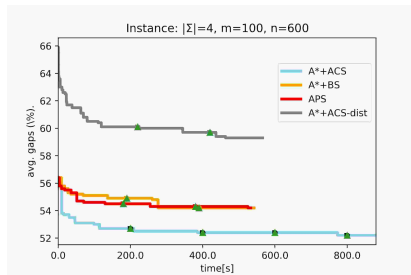
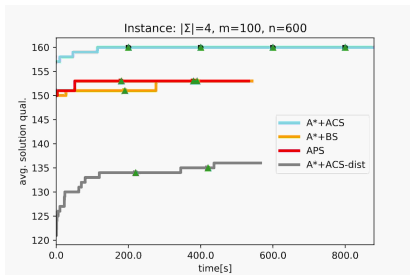


Figure: $m = 100, n = 600, |\Sigma|=4$.

- A^* proves optimality 106 instances of BL set
- APS and A^*+BS produce solutions and gaps of similar quality
- A^*+ACS ensures better quality of solutions / gaps for the middle-sized and hard-sized instance sets over time
- A^*+BS shows a slight advantage over A^*+ACS concerning quality of gaps on the smaller instances (more nodes expanded while statistically similar solutions obtained)
- A^*+BS and APS approaches require for more memory in comparison to the A^*+ACS

- A^* proves optimality 106 instances of BL set
- APS and A^*+BS produce solutions and gaps of similar quality
- A^*+ACS ensures better quality of solutions / gaps for the middle-sized and hard-sized instance sets over time
- A^*+BS shows a slight advantage over A^*+ACS concerning quality of gaps on the smaller instances (more nodes expanded while statistically similar solutions obtained)
- A^*+BS and APS approaches require for more memory in comparison to the A^*+ACS
- **TODO:** Algorithms have to be tuned

- The A^* search and two novel anytime approaches for LCS developed
- able to prove optimality of the small-sized instances (with up to $n = 100$ and $|\Sigma| \geq 12$) and return reasonable solutions and proven gaps for the middle and large-sized instances

- The A^* search and two novel anytime approaches for LCS developed
- able to prove optimality of the small-sized instances (with up to $n = 100$ and $|\Sigma| \geq 12$) and return reasonable solutions and proven gaps for the middle and large-sized instances

Future work concerns:

- improving gaps: find tighter upper bounds than UB_{\min}
- develop parallel implementations of the approaches

Thank you for your attention!