J Comb Optim (2008) 16: 127–154
DOI 10.1007/s10878-008-9139-z

A Lagrangian relaxation approach for the multiple sequence alignment problem

Ernst Althaus · Stefan Canzar

Published online: 5 March 2008

Abstract We present a branch-and-bound (bb) algorithm for the multiple sequence alignment problem (MSA), one of the most important problems in computational biology. The upper bound at each bb node is based on a Lagrangian relaxation of an integer linear programming formulation for MSA. Dualizing certain inequalities, the Lagrangian subproblem becomes a pairwise alignment problem, which can be solved efficiently by a dynamic programming approach. Due to a reformulation w.r.t. additionally introduced variables prior to relaxation we improve the convergence rate dramatically while at the same time being able to solve the Lagrangian problem efficiently. Our experiments show that our implementation, although preliminary, outperforms all exact algorithms for the multiple sequence alignment problem. Furthermore, the quality of the alignments is among the best computed so far.

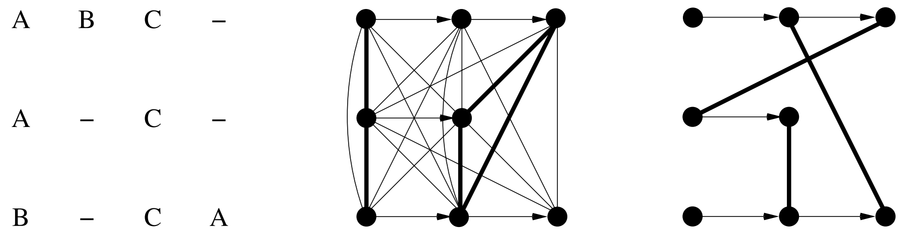
Keywords Sequence comparison · Lagrangian relaxation · Branch and bound

1 Introduction

Aligning DNA or protein sequences is one of the most important and predominant problems in computational molecular biology. Before we motivate this we introduce the following notation for the multiple sequence alignment problem.

Let $\mathcal{S} = \{s^1, s^2, \dots, s^k\}$ be a set of k strings over an alphabet Σ and let $\bar{\Sigma} = \Sigma \cup \{-\}$, where "$-$" (dash) is a symbol to represent "gaps" in strings. Given a string s, we let $\|s\|$ denote the number of characters in the string and s_l the lth character of s, for $l = 1, \dots, \|s\|$. We will assume that $\|s^i\| \geq 4$ for all strings s^i and let $n := \sum_{i=1}^{k} \|s^i\|$.

E. Althaus · S. Canzar (✉)
Max-Planck Institut für Informatik, Campus E14, 66123 Saarbrücken, Germany
e-mail: scanzar@mpi-sb.mpg.de



(a) Alignment of Input Sequences (b) Gapped Alignment Graph (c) Ordering Conflict

Fig. 1 (**a**) A possible alignment $\mathcal{A}$ of the input sequences $\mathcal{S} = \{ABC, AC, BCA\}$. (**b**) The gapped alignment graph for the sequences in $\mathcal{S}$. The thick edges specify alignment $\mathcal{A}$. (**c**) The alignment edges can not be realized at the same time in an alignment. Together with appropriate arcs of A_P, they form a mixed cycle

An *alignment* $\mathcal{A}$ of $\mathcal{S}$ is a set $\bar{\mathcal{S}} = \{\bar{s}^1, \bar{s}^2, \dots, \bar{s}^k\}$ of strings over the alphabet $\bar{\Sigma}$ where each string can be interpreted as a row of a two dimensional *alignment matrix*. The set $\bar{\mathcal{S}}$ of strings has to satisfy the following properties: (1) the strings in $\bar{\mathcal{S}}$ all have the same length, (2) ignoring dashes, string $\bar{s}^i$ is identical to string s^i, and (3) none of the columns of the alignment matrix is allowed to contain only dashes (see Fig. 1a).

If $\bar{s}^i_l$ and $\bar{s}^j_l$ are both different from "−", the corresponding characters in s^i and s^j are *aligned* and thus contribute a weight $w(\bar{s}^i_l, \bar{s}^j_l)$ to the value of $\mathcal{A}$. The pairwise scoring matrix w over the alphabet Σ models either costs or benefits, depending on whether we minimize distance or maximize similarity. In the following, we assume that we maximize the weight of the alignment. Moreover, a *gap* in s^j with respect to s^i is a maximal sequence $s^i_l\ s^i_{l+1}\ \dots\ s^i_m$ of characters in s^i that are aligned with dashes "−" in row j. Associated with each of these gaps is a cost. In the *affine gap cost* model the cost of a single gap of length q is given by the affine function $c_{\text{open}} + qc_{\text{ext}}$, i.e. such a gap contributes a weight of $-c_{\text{open}} - qc_{\text{ext}} = w_{\text{open}} + qw_{\text{ext}}$ to the total weight of the alignment. The problem calls for an alignment $\mathcal{A}$ whose overall weight is maximized.

Alignment programs still belong to the class of the most important Bioinformatics tools with a large number of applications. Pairwise alignments, for example, are mostly used to find strings in a database that share certain commonalities with a query sequence but which might not be known to be biologically related. Multiple alignments serve a different purpose. Indeed, they can be viewed as solving problems that are inverse to the ones addressed by pairwise string comparisons (Gusfield 1997). The inverse problem is to infer certain shared patterns from known biological relationships.

The question remains how a multiple alignment should be scored. The model that is used most consistently by far is the so called *sum of pairs* (SP) score. The SP score of a multiple alignment $\mathcal{A}$ is simply the sum of the scores of the pairwise alignments induced by $\mathcal{A}$ (Carrillo and Lipman 1988).

If the number k of sequences is fixed the multiple alignment problem for sequences of length n can be solved in time and space $\mathcal{O}(n^k)$ with (quasi)-affine gap costs (Gupta et al. 1995; Lermen and Reinert 2000; Reinert et al. 1997, 2000). More complex gap cost functions add a polylog factor to this complexity (Eppstein 1990; Larmore and Schieber 1990). However, if the number k of sequences is not fixed, Elias (2003) proved that multiple alignment with SP score is $\mathcal{NP}$-complete by a reduction from INDEPENDENT SET in 3-regular graphs (Garey and Johnson 1979). Hence it is unlikely that polynomial time algorithms exist and, depending on the problem size, various heuristics are applied to solve the problem approximately (see, e.g., (Notredame et al. 2000; Delcher et al. 1999; Subramanian et al. 2005)).

Althaus et al. (2002, 2006) propose a branch-and-cut algorithm for the multiple sequence alignment problem based on an integer linear programming (ILP) formulation. As solving the LP-relaxation is by far the most expensive part of the algorithm and even not possible for moderately large instances, we propose a Lagrangian approach to approximate the linear program and utilize the resulting bounds on the optimal value in a branch-and-bound framework. We assume that the reader is familiar with the Lagrangian relaxation approach to approximate linear programs.

The paper is organized as follows. In Sect. 2 we review the ILP formulation of the multiple sequence alignment problem, whose Lagrangian relaxation is described in Sect. 3. Our algorithm for solving the resulting problem is introduced in Sect. 4. Section 5 describes the approximation of the Lagrangian dual problem. Finally, computational experiments on a set of real-world instances are reported in Sect. 6.

2 Previous work

Althaus et al. (2006) use a formulation for the multiple sequence alignment problem as an ILP given by Reinert (1999).

For ease of notation, they define the *gapped alignment graph*, a mixed graph whose node set corresponds to the characters of the strings and whose edge set is partitioned into undirected alignment edges and directed positioning arcs as follows: $G = (V, E_A \cup A_P)$ with $V = V^i \cup \cdots \cup V^k$ and $V^i = \{u^i_j \mid 1 \le j \le \|s^i\|\}$, $E_A = \{uv \mid u \in V^i, v \in V^j, i \neq j\}$ and $A_P = \{(u^i_l, u^i_{l+1}) \mid 1 \le i \le k \text{ and } 1 \le l < \|s^i\|\}$ (see Fig. 1b). Furthermore, we denote with $\mathcal{G} = \{(u, v, j) \mid u, v \in V^i, j \neq i\}$ the set of all possible gaps.

An edge in E_A is *realized* by an alignment, if its endpoints are placed into the same column of the alignment matrix, i.e the corresponding characters are *aligned*. Accordingly, a gap (u^i_l, u^i_m, j) is *realized* by an alignment, if the substring of s^i from position l to position m is aligned to gap characters "$-$" in string s^j, whereas both s^i_{l-1} and s^i_{m+1} are aligned to characters in s^j. Arcs in A_p represent consecutivity of characters within the same string and are independent of the alignment.

In order to score the alignment, we assign each edge $u^i_l u^j_m \in E_A$ a weight $w_{u^i_l u^j_m} := w(s^i_l, s^j_m)$ and a gap (u^i_l, u^i_m, j) the weight $w_{(u^i_l, u^i_m, j)} := w_{\text{open}} + (m - l + 1) \cdot w_{\text{ext}}$, which represents the benefit of realizing that edge or gap.

We call pairs $(E', \mathcal{G}')$, for which there exists an alignment $\mathcal{A}$ such that E' and $\mathcal{G}'$ are the set of edges in E_A, respectively gaps in $\mathcal{G}$, that are realized by $\mathcal{A}$, *gapped traces*. Notice that different alignments might correspond to the same gapped trace $(E', \mathcal{G}')$, but all such alignments have the same score $\sum_{e \in E'} w_e + \sum_{g \in \mathcal{G}'} w_g$.

The ILP formulation uses a variable for every possible alignment edge $e \in E_A$, denoted by x_e, and one variable for every possible gap $g \in \mathcal{G}$, denoted by y_g. Reinert (1999) showed that the characteristic vectors of the gapped traces are exactly the $\{0, 1\}$-assignments to the variables such that

- (PaiwAl) we have pairwise alignments between every pair of strings,
- (MixedCy) there are no *mixed cycles*, i.e. in the subgraph of the gapped alignment graph consisting of the positioning arcs A_P and the realized edges $\{e \in E_A \mid x_e = 1\}$ there is no cycle that respects the direction of the arcs of A_p (and uses the edges of E_A in either direction) and contains at least one arc of A_P (see Fig. 1c),
- (Trans) transitivity is preserved, i.e. if u is aligned with v and v with w then u is aligned with w, for $u, v, w \in V$.

These three conditions are easily formulated as linear constraints (see Althaus et al. 2006). Given weights w_e associated with variables x_e, $e \in E_A$, and gap costs w_g associated with variables y_g, we denote the problem of finding a gapped trace (a solution satisfying (PaiwAl), (MixedCy) and (Trans)) which has the highest weight as (P) and its optimal value as $v(P)$. As the number of those inequalities is exponential Althaus et al. use a cutting plane framework to solve the LP relaxation (all inequalities have a polynomial separation algorithm). In their experiments they observed that the number of iterations in the cutting plane approach can be reduced, if they use additional variables $z_{(u,v)}$ for $u \in V^i$, $v \in V^j$, $i \neq j$, with the property that $z_{(u,v)} = 1$ iff at least one character of the string of u lying (not strictly) right of u is aligned to a character of the string of v lying (not strictly) left of v, i.e. $z_{(u_l^i, u_m^j)} = 1$, iff there is $l' \geq l$ and $m' \leq m$ with $x_{u_{l'}^i u_{m'}^j} = 1$. This condition is captured by the inequalities

$$\begin{aligned} 0 \leq z \leq 1, \qquad & z_{(u^i_{\|s^i\|}, u^j_1)} = x_{u^i_{\|s^i\|} u^j_1}, \\ & z_{(u^i_l, u^j_m)} \geq z_{(u^i_{l+1}, u^j_m)} + x_{u^i_l u^j_m} \quad \text{and} \\ & z_{(u^i_l, u^j_m)} \geq z_{(u^i_l, u^j_{m-1})} + x_{u^i_l u^j_m}. \end{aligned} \tag{1}$$

Notice that indicator variables x_e are associated with undirected edges $e = uv$, whereas variables z_e are associated with directed edges $e = (u, v)$. In the following, we describe the inequalities used in Althaus et al. (2006) to enforce (MixedCy). We refrain from explicitly specifying the inequalities enforcing (PaiwAl) and (Trans), as they are not crucial for the understanding of our approach.

Using these additional variables, we can define facets that guarantee (MixedCy) as follows. We model the mixed cycles as introduced above by letting $A_A = \{(u, v) \mid u \in V^i, v \in V^j, i \neq j\}$, i.e. for each undirected edge $uv \in E_A$, we have the two directed arcs (u, v) and (v, u) in A_A. Then a cycle $M \subseteq A_A \cup A_P$ in $(V, A_A \cup A_P)$ that contains at least one arc of A_P uniquely defines a mixed cycle. Where it is clear from the context, we therefore resign to distinguish between the term *mixed cycle* in its original meaning, namely cycles as defined in (MixedCy) having both undirected and directed edges, and their corresponding cycles in $(V, A_A \cup A_P)$, consisting exclusively of directed arcs. The set of all mixed cycles is denoted by $\mathcal{M}$. We show, that for a mixed cycle $M \in \mathcal{M}$ the inequality

$$\sum_{e \in M \cap A_A} z_e \leq |M \cap A_A| - 1 \tag{2}$$

is valid. Assume $\sum_{e \in M \cap A_A} z_e = |M \cap A_A|$. Consider an arbitrary arc $(u_l^i, u_m^j) \in M \cap A_A$. Since positioning arcs are directed "from left to right", the arc from A_A preceding (u_l^i, u_m^j) on cycle M must terminate in a node $u_{l'}^i$ with $l' \leq l$. Similarly, the arc from A_A succeeding (u_l^i, u_m^j) on cycle M must originate in a node $u_{m'}^j$ with $m' \geq m$. According to the definition of $z_{(u_l^i, u_m^j)}$, there is $l'' \geq l$ and $m'' \leq m$ with $x_{u_{l''}^i u_{m''}^j} = 1$. Thus by replacing every arc (u_l^i, u_m^j) from A_A on M by a (possibly empty) sequence of positioning arcs $(u_l^i, u_{l+1}^i), \ldots, (u_{l''-1}^i, u_{l''}^i)$, followed by edge $u_{l''}^i u_{m''}^j$, followed by a (possibly empty) sequence of positioning arcs $(u_{m''}^j, u_{m''+1}^j), \ldots, (u_{m-1}^j, u_m^j)$, we can construct a mixed cycle, all of which (undirected) alignment edges can not be realized at the same time and which thus represents a ordering conflict (see Fig. 1c). Therefore (2) must hold. These inequalities imply (MixedCy) as $z_{(u,v)} \geq x_{uv}$, and are called (*lifted*) *mixed cycle inequalities*. They define facets under appropriate technical conditions.

Assume a mixed cycle M contains at least two arcs of A_P and let (u_l^i, u_{l+1}^i) be one of them. Let M' be the cycle obtained from M by replacing arcs (v, u_l^i), (u_l^i, u_{l+1}^i), (u_{l+1}^i, w) (w.l.o.g. assume $w \neq u_{l+2}^i$) by arcs (v, u_l^i) and (u_l^i, w). Then the mixed cycle inequality for M' implies the mixed cycle inequality for M as $z_{(u_l^i, w)} \geq z_{(u_{l+1}^i, w)}$. In particular, a mixed cycle inequality can only define a facet if there is exactly one arc of A_P in M. The constraints (2) can be formulated similarly without using the additional z-variables.

3 Outline

Our Lagrangian approach is based on the integer linear program outlined above. Hence we have three classes of variables, X, Y and Z. Notice that a single variable x_{uv}, $y_{(u,v,j)}$, or $z_{(u,v)}$ involves exactly two sequences. Let $X^{i,j}$, $Y^{i,j}$, and $Z^{i,j}$ be the set of variables involving sequences i and j. If we restrict our attention to the variables in $X^{i,j}$, $Y^{i,j}$ and $Z^{i,j}$, for a specific pair of sequences i, j, a solution of the ILP yields a description of a pairwise alignment between sequences i and j, along with appropriate values for the variables in $Z^{i,j}$. The constraints (MixedCy) and (Trans) are used to guarantee that all pairwise alignments together form a multiple sequence alignment. We call an assignment of $\{0, 1\}$-values to variables in $(X^{i,j}, Y^{i,j}, Z^{i,j})$ such that $(X^{i,j}, Y^{i,j})$ imposes a pairwise alignment and $Z^{i,j}$ satisfies inequalities (1), an *extended pairwise alignment*. Given weights for the variables in $X^{i,j}$, $Y^{i,j}$ and $Z^{i,j}$, we call the problem of finding an extended pairwise alignment of maximum weight the *extended pairwise alignment problem*.

In our Lagrangian approach we dualize the constraints for condition (MixedCy) (i.e. inequalities (2)) and relax conditions (Trans) (during experiments it turned out that relaxing condition (Trans) is more efficient in practice as dualizing them). Hence our Lagrangian subproblem is an extended pairwise alignment problem. More precisely, if $\lambda_M \geq 0$ is the current multiplier for the mixed cycle inequality of $M \in \mathcal{M}$, we have to solve the Lagrangian relaxation problem

$$(LR_\lambda) \qquad \sum_{M\in\mathcal{M}} \lambda_M(|M \cap A_A| - 1) + \max \sum_{e\in E_A} w_e x_e + \sum_{g\in\mathcal{G}} w_g y_g - \sum_{M\in\mathcal{M}} \lambda_M \sum_{e\in M\cap A_A} z_e$$

$$\text{s.t.} \quad (X^{i,j}, Y^{i,j}, Z^{i,j}) \text{ forms an extended pairwise alignment for all } i, j.$$

We denote its optimal value with $v(LR_\lambda)$. As the number of inequalities that we dualize is exponential, we modify the subgradient method (SM) in a relax-and-cut fashion, as proposed by Fisher (1994) (see Sect. 5).

4 Solving the extended pairwise alignment problem

Recall how a pairwise alignment with gap cost is computed for two strings s and t of length n_s and n_t, respectively (without loss of generality we assume $n_t \leq n_s$). By a simple dynamic programming algorithm, we compute for every $1 \leq l \leq n_s$ and every $1 \leq m \leq n_t$ the optimal alignment of prefixes $s_1 \dots s_l$ and $t_1 \dots t_m$ that aligns s_l and t_m and whose score is denoted by $D(l, m)$. This can be done by comparing all optimal alignments for strings $s_1 \dots s_{l'}$ and $t_1 \dots t_{m'}$ for $l' < l$ and $m' < m$, adding the appropriate gap cost to the score $w(s_l, t_m)$ obtained for aligning s_l and t_m. If the weight of a gap is an arbitrary function $w(q)$ of its length q, the determination of the optimal alignment value $\max_{x\leq n_s, y\leq n_t}[D(x, y) + w(n_s - x) + w(n_t - y)]$, takes time $\mathcal{O}(n_s^2 n_t^2)$.[1]

In the affine gap weight model we can restrict the dependence of each cell in the dynamic programming matrix to adjacent entries in the matrix by associating more than one variable to each entry as follows. Besides computing $D(l, m)$, we compute the score of the optimal alignment of these substrings that aligns character s_l to a character t_k with $k < m$, denoted by $V(l, m)$, and the one that aligns t_m to a character s_k with $k < l$, denoted by $H(l, m)$. Hence, in a node $V(l, m)$, we have already paid the opening cost for the gap in t and we can traverse from $V(l, m)$ to $V(l, m + 1)$ by just adding w_{ext}, but not w_{open}. Each of the terms $D(l, m)$, $V(l, m)$ and $H(l, m)$ can be evaluated by a constant number of references to previously determined values and thus the running time reduces to $\mathcal{O}(n_s n_t)$.

The pairwise alignment problem can be interpreted as a longest path problem in an acyclic graph, having three nodes $D(l, m)$, $V(l, m)$ and $H(l, m)$ for every pair of characters $s_l \in s$, $t_m \in t$, referred to as cell (l, m). We call this graph the dynamic programming graph. In the further discussion we assume that nodes of a cell (l, m)

[1] The running time can be reduced to $\mathcal{O}(n_s^2 n_t)$ by distinguishing three different types of alignments (Sankoff and Kruskal 1983).

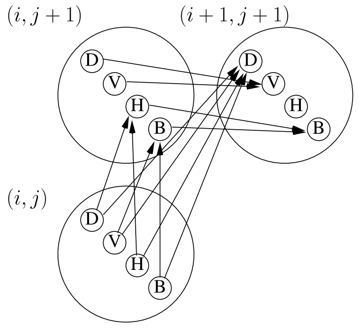


Fig. 2 Three cells of the dynamic programming matrix, with four values (nodes) associated to each of them describing the type of the alignment. The forth value $B(i, j)$ means that neither s_i is aligned to a character of t nor t_j is aligned to a character of s. Note that arcs (dependencies) are between certain values D, V, H and B, the target node determines the type of the partial alignment

are drawn at coordinates (l, m) in the plane. Furthermore, the term $\mathcal{S}(l, m)$, with $\mathcal{S} \in \{D, V, H\}$, is used interchangeably to refer both to a node in the dynamic programming graph and the score of the specific type of alignment it represents.

Each pairwise alignment corresponds to a path through this graph from $D(0, 0)$ to a node of the cell (n_s, n_t), with every arc of the path representing a certain kind of column in the alignment matrix, determined by the type of its target node (Fig. 2). An *alignment arc* from an arbitrary node in cell $(l-1, m-1)$ to node $D(l, m)$ corresponds to an alignment of characters s_l and t_m. Accordingly, a *gap arc* has a target node $V(l, m)$ or $H(l, m)$ and represents a gap opening (source node is $D(l, m-1)$ or $D(l-1, m)$, respectively) or a gap extension (source node is $V(l, m-1)$ or $H(l-1, m)$, respectively). We call gap arcs from a node of a cell (i, j) to a node of cell $(i, j+1)$ a *horizontal* arc, gap arcs from a node of a cell (i, j) to a node of cell $(i+1, j)$ a *vertical* arc, and alignment arcs *diagonal* arcs.

4.1 Simple algorithm

Now assume some variable $z_{(u,v)}$ is multiplied by a non-zero value in the objective function, as the arc $(u, v) \in A_A$ is contained in at least one mixed cycle M, to which a non-zero Lagrangian multiplier λ_M is associated. Recall that the multiplier of the variable $z_{(u,v)}$ in the objective function is $-\sum_{M \in \mathcal{M} | (u,v) \in M} \lambda_M$ (see (LR_λ)). Then we have to pay the multiplier as soon as our path traverses at least one alignment arc that enforces $z_{(u,v)} = 1$. Assume $s = s^i$, $t = s^j$, $u = u^i_l$ and $v = u^j_m$. Then $z_{(u,v)} = 1$, iff there is $l' \geq l$ and $m' \leq m$ such that $x_{u^i_{l'} u^j_{m'}} = 1$ (see definition of variables $z_{(u,v)}$ in (1)). In the dynamic program graph, this corresponds to alignment arcs whose target lies in the lower right rectangle from cell (l, m) (i.e. for the target $D(l', m')$ it holds that $l' \geq l$ and $m' \leq m$). Analogously, if u lies in string s^j and v in string s^i, this corresponds to alignment arcs whose target lies in an upper left rectangle. We call these rectangles *blue* and *red obstacles* and denote them by $\mathcal{O}_b(l, m)$ and $\mathcal{O}_r(l, m)$, respectively. Cell (l, m) is called the *origin* of the obstacle.

Let the set of all blue and red obstacles be denoted by $\mathcal{O}_b$ and $\mathcal{O}_r$, respectively, and let $\mathcal{O} = \mathcal{O}_b \cup \mathcal{O}_r$. Then the extended pairwise alignment problem is solvable by a dynamic program in $\mathcal{O}(n_s^2 n_t^2 |\mathcal{O}|)$ time, following the same approach as above: we compute the best alignment of all pairs of prefixes $s_1 \ldots s_l$ and $t_1 \ldots t_m$ that aligns s_l and t_m, based on all best alignments of strings $s_1 \ldots s_{l'}$ and $t_1 \ldots t_{m'}$, for $l' < l$ and

$m' < m$. We add the appropriate gap weight to $w(s_l, t_m)$ and subtract all multipliers that are associated with obstacles that have to be charged when aligning s_l and t_m but are not jet charged. This are the red obstacles with origin (i, j) with $l \leq i, m' < j \leq m$ and the blue obstacles with $m \leq j, l' < i \leq l$. Notice that the information that $s_{l'}$ and $t_{m'}$ are the last two aligned characters suffices to determine which multipliers we have to charge additionally when aligning s_l and t_m.

4.2 Improved algorithm

We reduce the complexity of the dynamic program by again decreasing the alignment's history, necessary to determine the benefit of any possible continuation in a partial alignment. The determination of the set of obstacles, whose associated penalty we have to pay when using an alignment arc, poses the major problem. For that we have to know the last alignment arc that has been used on our path. However, this arc can not be precomputed in a straightforward way, since the longest path in this context does not have optimal substructure.

We say that we *enter* an obstacle with an arc, if the target lies within the obstacle, but not the source. The key idea is to charge the Lagrangian multipliers as soon as we enter an obstacle no matter whether we enter it with an alignment arc (in which case we indeed have to charge the associated multiplier) or with a gap arc (such that we have to charge the associated multiplier only when using an alignment arc within the obstacle later). We introduce further nodes and edges which allow us to bypass obstacles in which we do not use any alignment arc.

When traversing an alignment arc with target $D(x, y)$, we charge the multipliers of all obstacles we enter, i.e. red obstacles with origin (x', y) for $x' \geq x$ and blue obstacles with origin (x, y') for $y' \geq y$. When traversing a gap arc we charge only multipliers of those obstacles we enter, in which we are still able to traverse an alignment arc (i.e. we do not charge the multiplier if the target of any alignment arc reachable does not lie in this obstacle). More precisely, for using the gap arc from a node in cell $(x - 1, y)$ to $H(x, y)$, we charge the multipliers of all blue obstacles having origin (x, y') with $y' > y$. Similarly, gap arcs from a node in cell $(x, y - 1)$ to $V(x, y)$ are charged. This motivates the following definition.

Definition 4.1 (Enclosing obstacles) The set of enclosing blue obstacles $\mathcal{Q}_b(p)$ of a cell $p = (x, y)$ contains all blue obstacles $\mathcal{O}_b(l, m)$ with $l \leq x, m > y$. Accordingly, $\mathcal{Q}_r(p) = \{\mathcal{O}_r(s, t) \mid s > x, t \leq y\}$. Furthermore we define $\mathcal{Q}(p) = \mathcal{Q}_b(p) \cup \mathcal{Q}_r(p)$.

Hence when using a gap arc, we charge multipliers of all obstacles *enclosing* the target but not the source. Notice that the set of obstacles enclosing a cell (x, y) contains exactly those obstacles whose associated multiplier we have to charge when using an alignment arc a from a node in cell (x, y) to $D(x + 1, y + 1)$, but which are not taken into account during the traversal of a.

The following simple facts are crucial for the understanding of our proofs.

- $\mathcal{Q}_b((x + 1, y)) \setminus \mathcal{Q}_b((x, y))$ is the set of obstacles whose associated Lagrangian multipliers we have to charge when using an arc from a node in cell (x, y) to node $H(x + 1, y)$.

- $\mathcal{Q}_b((x, y+1)) \setminus \mathcal{Q}_b((x, y)) = \emptyset$.
- For arbitrary cells p, q, r, it holds that $\{\mathcal{Q}(q) \setminus \mathcal{Q}(p)\} \cup \{\mathcal{Q}(r) \setminus \mathcal{Q}(q)\} \supseteq \mathcal{Q}(r) \setminus \mathcal{Q}(p)$.
- Consider $l' < l$ and $m' < m$. Let $\mathcal{A}$ be an alignment that aligns $s_{l'}$ with $t_{m'}$ and s_l with t_m with gaps in between. The set of obstacles whose multipliers have to be charged for the alignment of s_l and t_m contains obstacles in $\mathcal{Q}((l-1, m-1)) \setminus \mathcal{Q}((l', m'))$ plus obstacles entered with the alignment arc having target $D(l, m)$.

Again notice that we charge the multiplier of an obstacle at most once (when we enter the obstacle). Furthermore, we charge at least the multipliers we have to charge (the last two facts above), but we possibly charge more: we charge the multiplier of an obstacle o whenever we reach a cell enclosed by o, independent of the arc type. Hence, we have to ensure that we are able to bypass obstacles we do not have to pay, i.e. obstacles that are not enclosing any target node of an alignment arc traversed by the optimal path. We accomplish this by adding new nodes and arcs to the dynamic programming graph. Additionally we compute, for every pair of characters $s_l \in s$, $t_m \in t$, a fourth value $B(l, m)$ denoting the value of the optimal alignment that aligns either character s_l to "–" strictly left from t_m or character t_m to "–" strictly left from s_l, i.e. we have paid both opening costs. Hence every cell (l, m) contains a fourth node $B(l, m)$ in the dynamic programming graph.

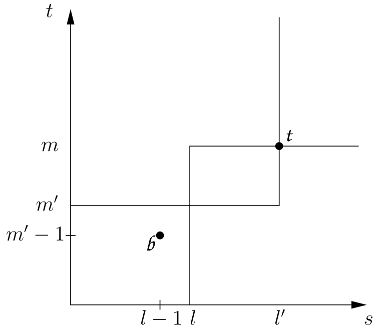
Before we introduce the new nodes and edges formally, we need some basic definitions. We call a pair of a blue obstacle $\mathcal{O}_b(l, m)$ and a red obstacle $\mathcal{O}_r(l', m')$ *conflicting*, if $l' \geq l$ and $m' \leq m$ (Fig. 3a). The *base* $\mathscr{b}(\mathcal{O}_b(l, m), \mathcal{O}_r(l', m'))$ of a pair of conflicting obstacles is defined as cell $(l-1, m'-1)$, the *tail* $\mathscr{t}(\mathcal{O}_b(l, m), \mathcal{O}_r(l', m'))$ as cell (l', m). We say a cell (l, m) *dominates* a cell (l', m'), denoted by $(l, m) < (l', m')$, if $l < l'$ and $m < m'$ ($(l, m) \leq (l', m')$, if $l \leq l'$ and $m \leq m'$). Accordingly, a blue (red) obstacle $\mathcal{O}_{b(r)}(l, m)$ *dominates* an obstacle $\mathcal{O}_{b(r)}(l', m')$, iff origin (l, m) dominates origin (l', m') (Fig. 3b). A blue (red) obstacle is *minimal* in set $\hat{\mathcal{O}}_b \subseteq \mathcal{O}_b$ ($\hat{\mathcal{O}}_r \subseteq \mathcal{O}_r$), if it is not dominated by any other obstacle in $\hat{\mathcal{O}}_b$ ($\hat{\mathcal{O}}_r$). We denote the set of obstacles that are dominated by a given obstacle o, by $\mathcal{D}(o)$.

Given a set $E' \subseteq E_A$ of alignment arcs, we call the set of obstacles we do not have to charge for using an edge in E' the *forbidden obstacles* w.r.t. E'.

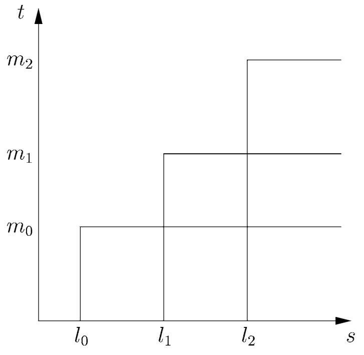
Assume that in the optimal extended alignment, (l, m) and $(l'+1, m'+1)$ with $l < l'$ and $m < m'$ are the targets of two realized alignment edges with gaps in between. We have to make sure that there is a path from $D(l, m)$ to the appropriate node of the cell (l', m') (e.g. B, if $l < l'$ and $m < l'$; H, if $l < l'$ and $m = m'$; V, if $l = l'$ and $m < m'$) which doesn't enter a forbidden obstacle. To achieve this, we try to proceed in the dynamic programming graph from $D(l, m)$ with gap arcs of either type until we can't go further without entering a forbidden obstacle. This motivates the addition of the B-node into a cell as it allows us to use gap arcs in either sequence alternatively without paying opening cost twice (see Fig. 3c).

Notice that we can not enter and leave an obstacle using exclusively gap arcs of one type (e.g. with horizontal arcs, we can enter blue obstacles, but not leave them). Hence we are able to reach a node of cell (l', m') without entering a forbidden obstacle, if we can reach a node of a cell (l', m'') with $m'' < m$. Analogously we can argue that we can reach the cell (l', m'), if we can reach a node of a cell (l'', m') with

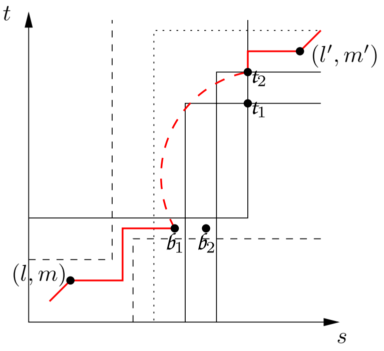
Fig. 3 (**a**) A pair of conflicting obstacles, together with its base $\mathscr{b}$ and its tail $\mathscr{t}$. (**b**) Obstacle $\mathcal{O}_b(l_0, m_0)$ dominates obstacles $\mathcal{O}_b(l_1, m_1)$ and $\mathcal{O}_b(l_2, m_2)$. Obstacle $\mathcal{O}_b(l_1, m_1)$ is minimal in $\mathcal{D}(\mathcal{O}_b(l_0, m_0))$. (**c**) We can bypass the *dashed* obstacles within the dp-graph, as they are not in conflict with any other obstacle. We can enter the *dotted* obstacle, as we have to subtract the multiplier then using the alignment arc with target $(l' + 1, m' + 1)$. Hence, we can reach $\mathscr{b}_1$ from (l, m) within the dp-graph, jump to $\mathscr{t}_2$, and proceed in the dp-graph to (l', m')



(a) Conflicting obstacles



(b) Dominating obstacles



(c) Path construction

$l'' < l'$. In particular, we are able to reach the appropriate node of the cell (l', m') within the dynamic programming graph, if the type of the cell is D, H, or V.

Hence we only consider the case where we want to reach the B-node of the cell (l', m') and cannot proceed from a node of the cell (l_b, m_b) with $l_b < l'$ and $m_b < m'$. In this case, (l_b, m_b) is the base of a pair of conflicting forbidden obstacles. More precisely, l_b is the smallest value such that there is a pair of forbidden conflicting obstacles with base (l_b, m''). Similarly, m_b is the smallest value such that there is a base (l'', m_b) of a pair of conflicting forbidden obstacles. Hence (l_b, m_b) dominates the base of every pair of forbidden conflicting obstacles.

Analogously, we can argue that there is a tail (l_t, m_t) of a pair of conflicting obstacles from which we can reach $B(l', m')$ without entering a forbidden obstacle (if we are not able to reach the cell (l', m') within the dynamic programming graph) and that (l_t, m_t) dominates (l', m'). Furthermore the base (l_b, m_b) dominates the tail (l_t, m_t), what we can see as follows. Let (o_b, o_r) be any pair of forbidden conflicting obstacles. The cell (l_b, m_b) dominates the base of (o_b, o_r). This base dominates the tail of (o_b, o_r), which again dominates (l_t, m_t).

Therefore, the insertion of arcs from the four nodes of the base of every pair of conflicting obstacles (o_b, o_r) to the B-node of the target of every pair (o'_b, o'_r) such that $\mathcal{b}(o_b, o_r) \leq \mathcal{b}(o'_b, o'_r)$ and $\mathcal{t}(o_b, o_r) \leq \mathcal{t}(o'_b, o'_r)$, would enable us to "jump over" obstacles that we do not have to pay. The weights for these arcs are determined by the cost of the gaps leading from $\mathcal{b}(o_b, o_r)$ to $\mathcal{t}(o'_b, o'_r)$ plus the penalties implied by obstacles enclosing $\mathcal{t}(o'_b, o'_r)$, but not $\mathcal{b}(o_b, o_r)$.

As the number of conflicting obstacles is at most $|\mathcal{O}|^2$, the number of additional arcs is at most $\mathcal{O}(|\mathcal{O}|^4)$ and hence the running time is $\mathcal{O}(n_s n_t + |\mathcal{O}|^4)$.

4.2.1 The bypass graph

To further reduce the number of additional arcs (dependencies) in our dynamic programming graph, we introduce the *bypass graph*, which is correlated to the transitive reduction of the induced subgraph on the set of newly added arcs.

The nodes of the bypass graph (formal definition below) represent pairs of conflicting obstacles. Intuitively, reaching a node v in the bypass graph along a path p, with $\mathcal{b}(v) = (l, m)$ and $\mathcal{t}(v) = (l', m')$, can be interpreted as having a consecutive run of alternate gaps $s_{g+1}, \ldots, s_{l'}$ and $t_{h+1}, \ldots, t_{m'}$, with $g \leq l$ and $h \leq m$. In particular, each node in the bypass graph (bpg) is connected to the B-node of its corresponding tail via an edge of weight 0. Note that in this case, the last alignment arc on path p has target node $D(g, h)$.

For cells (i, j) and (l, m) let $\|(i, j) - (l, m)\|_1$ denote Minkowski's $\mathcal{L}_1$ distance between points (i, j) and (l, m) in the plane. An outgoing edge (v, w) to another bpg node w models the extension of a gap in one of the strings by $\|\mathcal{t}(w) - \mathcal{t}(v)\|_1$ characters, i.e. by traversing an edge in the bpg the cell of the tail moves vertically upwards or horizontally to the right. Contrariwise, the cell of the base moves simultaneously to the right, respectively upwards. Therefore, the weight of a bpg edge (v, w) is composed of

- the cost of extending a gap by $\|\mathcal{t}(w) - \mathcal{t}(v)\|_1$ characters,
- minus the sum of multipliers associated with obstacles enclosing w but not v,
- the sum of multipliers associated with obstacles that we are leaving when proceeding from $\mathcal{t}(v)$ to $\mathcal{t}(w)$ and which do not enclose the target node of any alignment arc on our path.

The third part is based on the fact, that the structure of our overall graph prevents any path from reentering an already left obstacle and thus the remaining part of path p going from $\mathcal{t}(w)$ to (n_s, n_t) cannot traverse an alignment arc whose target node is

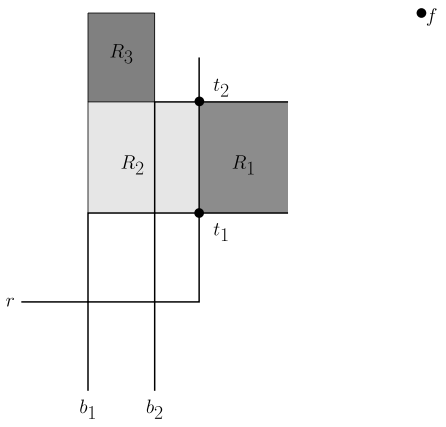


Fig. 4 Assume we traverse an edge of the bypass graph from the node corresponding to the pair of conflicting obstacles (b_1, r) to the pair (b_2, r) and we will leave the bypass graph at a node whose target is at f. The red obstacles whose origin lie in R_1 are entered when proceeding from the tail t_1 of (b, r_1) to the tail t_2 of (b, r_2) and hence the corresponding multipliers have to be subtracted. Furthermore, no blue obstacles are entered. At the same time blue obstacles that originate in area R_2 are left. Since the last alignment arc is "to the left" of base $\not b(b, r_1)$ those obstacles do not enclose any alignment arc and we therefore add their associated multipliers. We do not add multipliers of any red obstacle. After reaching node (b, r_2) we have to weaken our information about the last alignment arc to be "to the left" of $\not b(b, r_1)$. As a consequence, we would not be able to decide whether we have to pay for obstacles originating in R_3 or not. Hence we must have edges in the bypass graph that enable us to reach f on a path, on which these regions do not contain any origins of obstacles

enclosed by such an obstacle. As it concerns the determination of whether the part of path p going from $(0, 0)$ to $t(w)$ traverses such an alignment arc, note that the weight of a bpg edge can incorporate only "local" information. Specifically, we have to deal with the inexact information that the last alignment arc is "to the lower left" of $\not b(v)$. Similarly, when continuing from node w, we must not make any assumptions about the node from where we reached w and therefore we will have to further weaken our knowledge about the position of the last alignment arc to that it is to the lower left of $\not b(w)$. As a consequence, in the third part only those among the obstacles we are leaving can be considered, that are not enclosing $\not b(v)$. This is illustrated in Fig. 4. As we will see below, the definition of the edge set ensures the there always exists of a path through the bpg along which this set of obstacles is equal to the set of obstacles that are not enclosing the target node of any alignment arc lying on the path (compare third part).

Definition 4.2 (Bypass graph) We define the *Bypass Graph* (*bpg*) $G = (\mathcal{V}, \mathcal{E}, l)$ with edge set $\mathcal{E} \subset \mathcal{V} \times \mathcal{V}$ and length function $l \colon \mathcal{E} \to \mathbb{R}$ as follows. The vertex set $\mathcal{V}$ contains all pairs v of conflicting obstacles. Let v^b and v^r denote the blue and red obstacle of v, respectively. $\mathcal{E} = \mathcal{E}_b \cup \mathcal{E}_r$, where $\mathcal{E}_b = \{(v, w) \mid v^r = w^r$ and w^b is minimal in $\mathcal{D}(v^b)\}$ and $\mathcal{E}_r = \{(v, w) \mid v^b = w^b$ and w^r is minimal in $\mathcal{D}(v^r)\}$. Every edge $(v, w) \in \mathcal{E}_\kappa$, $\kappa \in \{b, r\}$, is assigned a length $l((v, w)) = w_{\text{ext}} \cdot \|t(w) - t(v)\|_1 - \sum_{o \in \mathcal{Q}^-(v,w)} \lambda(o) + \sum_{o \in \mathcal{Q}^+(v,w)} \lambda(o)$, where $\mathcal{Q}^-(v, w) =$

$\mathcal{Q}(t(w)) \setminus \mathcal{Q}(t(v))$ and $\mathcal{Q}^+(v,w) = \{\mathcal{O}_\kappa(i,j) \in \mathcal{Q}_\kappa(t(v)) \setminus \mathcal{Q}_\kappa(t(w)) \mid w^\kappa = \mathcal{O}_\kappa(l,m), i \geq l, j \leq m\}$.

We connect the bypass graph to the dynamic programming graph by arcs as follows: If (i,j) is the base of a pair of conflicting obstacles with corresponding node $v \in V$ in the bpg we add arcs of all nodes in cell (i,j) to v (recursion formula (8)) and by arcs from all $v \in \mathcal{V}$ to the B-node of their tail $t(v)$ (formula (7)).

The overall structure of the resulting graph, whose longest path from a dedicated start node in cell (0, 0) to a node in cell (n_s, n_t) corresponds to the optimal pairwise alignment, can be described in terms of the following recurrences (base case omitted):

$$C(l,m) = \max\{D(l,m), V(l,m), H(l,m), B(l,m)\} \tag{3}$$

with

$$D(l,m) = C(l-1,m-1) + w(s_l,t_m) - \sum_{o=\mathcal{O}_r(i,m), i\geq l} \lambda(o) - \sum_{o=\mathcal{O}_b(l,j), j\geq m} \lambda(o) \tag{4}$$

$$V(l,m) = \max\left\{\begin{matrix} D(l,m-1) + w_{\text{ext}} + w_{\text{open}} \\ V(l,m-1) + w_{\text{ext}} \end{matrix}\right\} - \sum_{o=\mathcal{O}_r(i,m), i>l} \lambda(o) \tag{5}$$

$$H(l,m) = \max\left\{\begin{matrix} D(l-1,m) + w_{\text{ext}} + w_{\text{open}} \\ H(l-1,m) + w_{\text{ext}} \end{matrix}\right\} - \sum_{o=\mathcal{O}_b(l,j), j>m} \lambda(o) \tag{6}$$

$$B(l,m) = \max\left\{\begin{array}{l} \max_{v\in\mathcal{V}: t(v)=(l,m)}\{\delta(v)\} \\ B(l-1,m) + w_{\text{ext}} - \sum_{o=\mathcal{O}_b(l,j), j>m} \lambda(o) \\ B(l,m-1) + w_{\text{ext}} - \sum_{o=\mathcal{O}_r(i,m), i>l} \lambda(o) \\ V(l-1,m) + w_{\text{ext}} + w_{\text{open}} - \sum_{o=\mathcal{O}_b(l,j), j>m} \lambda(o) \\ H(l,m-1) + w_{\text{ext}} + w_{\text{open}} - \sum_{o=\mathcal{O}_r(i,m), i>l} \lambda(o) \end{array}\right. \tag{7}$$

where

$$\delta(v) = \max\left\{\begin{array}{l} \max_{u:(u,v)\in\mathcal{E}}\{\delta(u) + l((u,v))\} \\ \left\{\begin{array}{l} D(\mathscr{b}(v)) + q w_{\text{ext}} + 2w_{\text{open}} \\ V(\mathscr{b}(v)) + q w_{\text{ext}} + w_{\text{open}} \\ H(\mathscr{b}(v)) + q w_{\text{ext}} + w_{\text{open}} \\ B(\mathscr{b}(v)) + q w_{\text{ext}} \end{array}\right\} - \sum_{o\in\{\mathcal{Q}(t(v))\setminus\mathcal{Q}(\mathscr{b}(v))\}} \lambda(o) \end{array}\right. \tag{8}$$

with q being the Manhattan distance between the tail and the base of node v, i.e. $q = \|t(v) - \mathscr{b}(v)\|_1$.

4.3 Complexity

Obviously there are at most $|\mathcal{O}|^2$ conflicting pairs of obstacles and hence the number of additional nodes $|\mathcal{V}|$ is at most $|\mathcal{O}|^2$. From Definition 4.2 it follows immediately that the number of additional arcs $|\mathcal{A}|$ is at most $\mathcal{O}(|\mathcal{O}|^3)$, as an edge of the bypass graph is defined by three obstacles. Therefore the running time to compute an optimal solution to the extended pairwise alignment problem is $\mathcal{O}(nm + |\mathcal{O}|^3)$.

We improve the practical performance of our algorithm for solving the extended pairwise alignment problem by applying an A^*-approach: Notice that the scores $D(l,m)$, $V(l,m)$, $H(l,m)$ and $B(l,m)$ during an iteration of the subgradient optimization (see Sect. 5) can be at most the scores of the first iteration, i.e. when all multipliers λ are set to 0. Therefore, the length of a longest path from any node (l,m) to (n_s, n_t) determined in the first iteration provides a heuristic estimate for all other iterations, which is monotonic and thus the first path found from $(0,0)$ to (n_s, n_t) is optimal.

4.4 Correctness

Let p be a path starting at $D(0,0)$ through the dp-graph and the bypass graph ending at a node $D(i,j)$. Furthermore, let $D(l,m)$ be the last node of type D on p preceding $D(i,j)$ and let $\hat{p}$ be the prefix of p up to node $D(l,m)$. If d denotes the score of the alignment of prefixes $s_1 \dots s_l$ and $t_1 \dots t_m$ induced by $\hat{p}$, the score of the alignment induced by p is $d + w(s_i, t_j) + w_{\text{ext}} \cdot \|(i-1, j-1) - (l,m)\|_1 + r \cdot w_{\text{open}} - \sum_{o=\mathcal{O}_r(i',j') | i \le i', m < j' \le j} \lambda(o) - \sum_{o=\mathcal{O}_b(i',j') | j \le j', l < i' \le i} \lambda(o)$, where $r = 0$, if $(i-1, j-1) = (l,m)$, $r = 2$, if $i - 1 > l$ and $j - 1 > m$ and $r = 1$ otherwise.

Theorem 4.3 *Given strings s and t of length n_s and n_t, respectively, $D(x,y)$, for $1 \le x \le n_s$ and $1 \le y \le n_t$, is equal to the value of an optimal extended pairwise alignment of prefixes $s_1 \dots s_x$ and $t_1 \dots t_y$ that aligns s_x with t_y.*

Hence the optimal extended pairwise alignment of s and t can be determined by iterating over all $D(x,y)$, $1 \le x \le n_s$ and $1 \le y \le n_t$, adding the appropriate weight for the remaining gaps. Alternatively, it can be easily seen that the value of the optimal extended pairwise alignment of s and t corresponds to the maximum of $D(n_s, n_t)$, $V(n_s, n_t)$, $H(n_s, n_t)$, and $B(n_s, n_t)$.

Proof Consider arbitrary but fixed indices $1 \le l < l' < n_s$ and $1 \le m < m' < n_t$ and assume that $D(l,m)$ and $D(l'+1, m'+1)$ are the targets of two realized alignment edges with gaps in between. We will show in Lemmas 4.5 and 4.6 that

(a) there is a path of length

$$w_{\text{ext}} \cdot \|(l', m') - (l,m)\|_1 + 2 \cdot w_{\text{open}} - \sum_{o \in \{\mathcal{Q}((l',m')) \setminus \mathcal{Q}((l,m))\}} \lambda(o)$$

between $D(l,m)$ and $B(l', m')$,

(b) any path between $D(l,m)$ and $B(l',m')$ that does not traverse any alignment arc has length at most

$$w_{\text{ext}} \cdot \|(l', m') - (l,m)\|_1 + 2 \cdot w_{\text{open}} - \sum_{o \in \{\mathcal{Q}((l',m')) \setminus \mathcal{Q}((l,m))\}} \lambda(o).$$

Similar assumptions can be made if $l = l'$ or $m = m'$, where we pay the gap opening cost at most once and the path ends at a node of type H, V or D.

Using these two facts, we can prove the statement of the theorem by induction over x and y. For $x = 0$ and $y = 0$ there is nothing to show. Consider $x, y > 0$.

Assume in the optimal extended pairwise alignment that aligns s_x and t_y the last alignment arc preceding the one with target $D(x, y)$ has target $D(l, m)$. Using fact (a) and the induction hypothesis, we obtain, by setting $q := \|(x-1, y-1) - (l, m)\|_1$ and using r as defined above,

$$\begin{aligned} D(x, y) &\geq D(l, m) + q \cdot w_{\text{ext}} + r \cdot w_{\text{open}} - \sum_{o \in \{\mathcal{Q}((x-1,y-1)) \setminus \mathcal{Q}((l,m))\}} \lambda(o) \\ &\quad + w(s_x, t_y) - \sum_{o = \mathcal{O}_r(i,y), i \geq x} \lambda(o) - \sum_{o = \mathcal{O}_b(x,j), j \geq y} \lambda(o) \qquad (9) \\ &= D(l, m) + q \cdot w_{\text{ext}} + r \cdot w_{\text{open}} - \sum_{o \in \{\hat{\mathcal{O}}_r \cup \hat{\mathcal{O}}_b\}} \lambda(o), \qquad (10) \end{aligned}$$

where $\hat{\mathcal{O}}_r = \{\mathcal{O}_r(i, j) \mid x \leq i, m < j \leq y\}$ and $\hat{\mathcal{O}}_b = \{\mathcal{O}_b(i, j) \mid y \leq j, l < i \leq x\}$. This value is equal to the value of the optimal extended pairwise alignment of prefixes $s_1 \dots s_x$ and $t_1 \dots t_y$ that aligns s_x and t_y.

Now let p be the longest path ending in $D(x, y)$. Notice that the last arc of path p is an alignment arc. Let $D(l, m)$ be the target of the last alignment arc of p preceding $D(x, y)$. Using fact (b) and the induction hypothesis, we can simply replace "$\geq$" in (9) by "$\leq$" to obtain analogously

$$D(x, y) \leq D(l, m) + q \cdot w_{\text{ext}} + r \cdot w_{\text{open}} - \sum_{o \in \{\hat{\mathcal{O}}_r \cup \hat{\mathcal{O}}_b\}} \lambda(o), \qquad (11)$$

where q, r, $\hat{\mathcal{O}}_r$ and $\hat{\mathcal{O}}_r$ are as defined above. This value corresponds to the value of the extended pairwise alignment of prefixes $s_1 \dots s_x$ and $t_1 \dots t_y$ that aligns s_x with t_y and s_l with t_m. Furthermore, it is based on the optimal extended pairwise alignment of prefixes $s_1 \dots s_l$ and $t_1 \dots t_m$ that aligns s_x with t_y. Clearly, the score of this specific alignment is bounded from above by the value of the optimal extended alignment of prefixes $s_1 \dots s_x$ and $t_1 \dots t_y$ that aligns s_x with t_y. $\square$

It remains to show assumptions (a) and (b) and their modifications for the case $l' = l$ or $m' = m$ used in the proof. If $l' = l$ and $m' = m$, there is nothing to show. If $l' = l$ or $m' = m$ and the other inequality is strict, we exclusively use gap arcs in one string and we therefore do not leave obstacles that we enter. Hence, we do not need to enter the bypass graph and can proceed simply in the dp-graph. It remains to show the facts for the case $l' < l$ and $m' < m$. Fact (a) mainly relies on the existence of a path through the bypass graph that represents a consecutive run of alternate gaps in either string and that is penalized only by multipliers assigned to newly entered obstacles:

Lemma 4.4 *Given a node $v \in \mathcal{V}$ of the bpg, $\mathcal{b}(v) = (l_\mathcal{b}, m_\mathcal{b})$, and a cell (l', m') with $t(v) < (l', m')$, there exists a node $v_n \in \mathcal{V}$ and path p through the bpg from*

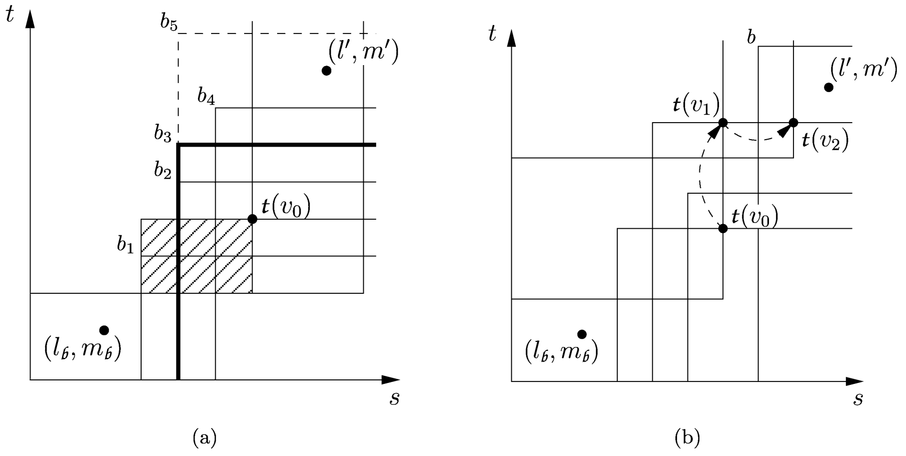


Fig. 5 Given base $(l_б, m_б)$ and a cell (l', m') as in Lemma 4.4. (**a**) The initial node v_0 in the bpg is determined by the *shaded rectangle*. Note that $\psi(v_0^b) > \psi(b_1)$. In the example, set Q_b^0 contains blue obstacles b_2, b_3, b_4, but not b_5, since b_5 encloses (l', m'). b_3 is a leftmost obstacle in Q_b^0 (min. property) and $\psi(b_3) > \psi(b_2)$ (max. property) and therefore $v_1 = (b_3, v_0^r)$. (**b**) An example sequence $\langle v_i \rangle_0^2$ depicted by the sequence of its tail cells, $(v_0, v_1) \in \mathcal{E}_b$, $(v_1, v_2) \in \mathcal{E}_r$. The sequence terminates at v_2 as b encloses (l', m')

every source node $\mathcal{S}(l_б, m_б)$, $\mathcal{S} \in \{D, V, H, B\}$, to the node $B(t(v_n))$ of length $w_{\text{ext}} \cdot ||t(v_n) - б(v)||_1 + r \cdot w_{\text{open}} - \sum_{o \in \{Q(t(v_n)) \setminus Q(б(v))\}} \lambda(o)$, where $r = 2$ if $\mathcal{S} = D$, $r = 1$ if $\mathcal{S} \in \{H, V\}$ and $r = 0$ if $\mathcal{S} = B$, such that $\{Q(t(v_n)) \setminus Q(б(v))\} \subseteq Q((l', m'))$, i.e. obstacles enclosing $t(v_n)$ but not $б(v)$ also enclose (l', m').

In the following proof of Lemma 4.4 we use functions ξ and ψ that are defined for an obstacle $o = \mathcal{O}_\kappa(l, m)$ as $\xi(o) = l$ and $\psi(o) = m$. Furthermore, we denote by $A \uplus B$ the union of disjoint sets A and B.

Proof We construct a sequence $\langle v_i \rangle_0^n$ of pairs of conflicting obstacles as follows (compare Fig. 5a): We select $v_0 = (v_0^b, v_0^r)$ with maximal $\xi(v_0^r)$ and $\psi(v_0^b)$, such that $б(v_0) = (l_б, m_б)$ and $v_0^b, v_0^r \notin Q((l', m'))$. Let Q_b^i be the set of blue obstacles that enclose the tail of pair v_i but neither cell (l', m') nor $(l_б, m_б)$, i.e. $Q_b^i = Q_b(t(v_i)) \setminus (Q_b(б(v_0)) \cup Q_b((l', m')))$. Accordingly, $Q_r^i = Q_r(t(v_i)) \setminus \{Q_r(б(v_0)) \cup Q_r((l', m'))\}$. Then for $i \geq 1$, if $Q_b^{i-1} \neq \emptyset$, v_i is obtained from v_{i-1} by picking the uppermost among the leftmost blue obstacles in Q_b^{i-1} while keeping the red obstacle unchanged, i.e. $v_i = (\mathcal{O}_b(g, h), v_{i-1}^r)$, with $\mathcal{O}_b(g, h) \in Q_b^{i-1}$ such that $\forall g', h', \mathcal{O}_b(g', h') \in Q_b^{i-1} : g \leq g'$ (min. property) and $\forall h', \mathcal{O}_b(g, h') \in Q_b^{i-1} : h > h'$ (max. property). Similarly, if $Q_b^{i-1} = \emptyset$ but $Q_r^{i-1} \neq \emptyset$, we retain the blue obstacle and choose the rightmost among the lowermost red obstacles in Q_r^{i-1}, i.e. we set $v_i = (v_{i-1}^b, \mathcal{O}_r(g, h))$, with $\mathcal{O}_r(g, h) \in Q_r^{i-1}$ such that $\forall g', h', \mathcal{O}_r(g', h') \in Q_r^{i-1} : h' \geq h$ and $\forall g', \mathcal{O}_r(g', h) \in Q_r^{i-1} : g' < g$. The sequence terminates at v_n, if $Q_b^n = Q_r^n = \emptyset$ (Fig. 5b).

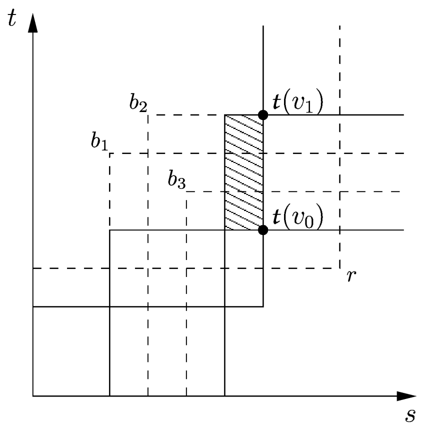


Fig. 6 Consider $(v_0, v_1) \in \mathcal{E}_b$ lying on a path $\hat{p}$ through the bpg as described in Lemma 4.4. Nodes v_0, v_1 are represented by their corresponding tails and by obstacles drawn by *solid lines*. Obstacles in $\mathcal{Q}^+(v_0, v_1)$ originate in the *shaded rectangle*. The existence of obstacle b_1 is in contradiction to the maximality of $\psi(v_0^b)$, b_2 is in conflict with the min. property of $\xi(v_1^b)$. According to the definition of an edge in the bpg, v_1^b is minimal in $\mathcal{D}(v_0^b)$ and therefore obstacle b_3 cannot exist. It follows $\mathcal{Q}^+(v_0, v_1) = \mathcal{Q}^+(1)$

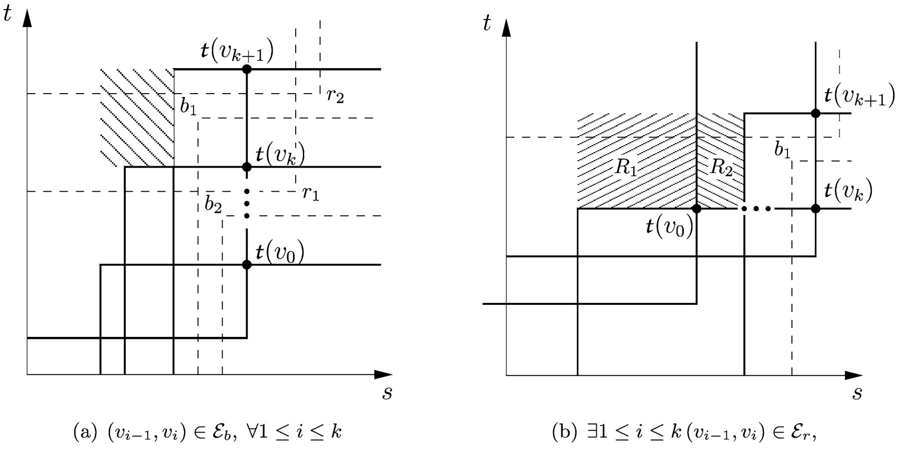
In the following we show that nodes in the bypass graph representing pairs of conflicting obstacles in $\langle v_i \rangle_0^n$ lie on a path $\hat{p}$ that can be easily extended to a path p having the required properties. It can be easily verified that there exists an edge between nodes corresponding to two consecutive pairs of obstacles in $\langle v_i \rangle_0^n$: the min. and max. properties of our construction of sequence $\langle v_i \rangle_0^n$ ensure $v_i^b \in \mathcal{D}(v_{i-1}^b)$ if $\mathcal{Q}_b^{i-1} \neq \emptyset$, and $v_i^r \in \mathcal{D}(v_{i-1}^r)$ otherwise. Furthermore, the existence of an obstacle $\hat{v}^\kappa \in \mathcal{D}(v_{i-1}^\kappa)$ with $\hat{v}^\kappa < v_i^\kappa$ would be in contradiction to the min. property of v_i^κ, meaning v_i^κ is minimal in $\mathcal{D}(v_{i-1}^\kappa)$ and thus $(v_{i-1}, v_i) \in \mathcal{E}_\kappa$, for all $1 \le i \le n$, $\kappa \in \{b, r\}$.

We will argue by induction on the number of edges k, $1 \le k \le n$, on a prefix of the path induced by sequence $\langle v_i \rangle_0^n$, that

$$\sum_{i=1}^{k} l(v_{i-1}, v_i) = w_{\text{ext}} \cdot ||t(v_k) - t(v_0)||_1 - \sum_{o \in \mathcal{Q}^-(k)} \lambda(o) + \sum_{o \in \mathcal{Q}^+(k)} \lambda(o), \tag{12}$$

with $\mathcal{Q}^-(k) = \mathcal{Q}(t(v_k)) \setminus \mathcal{Q}(t(v_0))$ and $\mathcal{Q}^+(k) = \mathcal{Q}(t(v_0)) \setminus \{\mathcal{Q}(\mathit{b}(v_0)) \cup \mathcal{Q}(t(v_k))\}$. In other words, the length of path $\hat{p}$, going from v_0 to v_k, accounts for the extension cost of gaps between cells $t(v_0)$ and $t(v_k)$ and is penalized by Lagrangian multipliers associated with obstacles enclosing $t(v_k)$ but not $t(v_0)$. Additionally, penalties of obstacles that $\hat{p}$ leaves are recovered, if they enclose $t(v_0)$ but not $\mathit{b}(v_0)$. Note that these obstacles are being paid for when traversing an arc connecting a node in cell $\mathit{b}(v_0)$ of the dynamic programming graph with bpg node v_0. Also, the weight of this arc incorporates any gap opening costs, depending on the type of its source node. Crucial in this context is, that multipliers assigned to obstacles that $\hat{p}$ enters along one arc and leaves along a later arc cancel out each other.

For the base case ($k = 1$) it suffices to show that $\mathcal{Q}^+(1) = \mathcal{Q}^+(v_0, v_1)$ (compare (12) for $k = 1$ with the length of an edge in the bpg, Definition 4.2). W.l.o.g. assume $(v_0, v_1) \in \mathcal{E}_b$ (see Fig. 6). Note that for general $(v_{i-1}, v_i) \in \mathcal{E}_b$ every red obstacle enclosing $t(v_{i-1})$ also encloses $t(v_i)$ (e.g. red obstacle r in Fig. 6) and thus $\mathcal{Q}(t(v_{i-1})) \setminus \mathcal{Q}(t(v_i)) \subseteq \mathcal{O}_b$. For every element $o \in \mathcal{Q}^+(v_0, v_1)$ it holds



t
$t(v_{k+1})$
r_2
b_1
$t(v_k)$
r_1
b_2
$t(v_0)$
s
(a) $(v_{i-1}, v_i) \in \mathcal{E}_b, \forall 1 \le i \le k$
t
$t(v_{k+1})$
b_1
R_1
R_2
$t(v_k)$
$t(v_0)$
s
(b) $\exists 1 \le i \le k\, (v_{i-1}, v_i) \in \mathcal{E}_r$,

Fig. 7 (**a**) Obstacles in $\mathcal{Q}^-(k)$ are from $\mathcal{O}_r$ and enclose $t(v_{k+1})$, see obstacle r_1. Therefore $\mathcal{Q}^-(k+1)$ is obtained by simply adding obstacles that enclose $t(v_{k+1})$, but not $t(v_k)$, like obstacle r_2. Obstacles in $\mathcal{Q}^+(k+1)$ can be divided into two subsets, depending on whether they enclose $t(v_k)$ (obstacle b_1) or not (obstacle b_2). The latter one coincides with set $\mathcal{Q}^+(k)$. The first subset is equal to set $\mathcal{Q}^+(v_k, v_{k+1})$, as the min. and max. properties of elements of sequence $\langle v_i \rangle$ imply the *shaded rectangle* to be empty. (**b**) Obstacles in $\mathcal{Q}^-(k)$ must not enclose $t(v_{k+1})$ (e.g. obstacle b_1) and thus have to be removed from $\mathcal{Q}^-(k) \uplus \mathcal{Q}^-(v_k, v_{k+1})$ to obtain $\mathcal{Q}^+(k+1)$. Note that no blue obstacle originates in rectangles R_1 (an edge in $\mathcal{E}_r$ is traversed only if there is no outgoing edge in $\mathcal{E}_b$) or R_2 (min. and max. properties of elements of $\langle v_i \rangle$). Therefore obstacles enclosing $t(v_k)$ but not $t(v_{k+1})$ do not enclose $t(v_0)$ and $\mathcal{Q}^+(k+1) = \mathcal{Q}^+(k)$ follows (obstacles enclosing $t(v_0)$ but not $t(v_k)$ do not enclose $t(v_{k+1})$)

$o \notin \mathcal{Q}(b(v_0))$ and $o \notin \mathcal{Q}(t(v_1))$ by definition, and thus $\mathcal{Q}^+(v_0, v_1) \subseteq \mathcal{Q}^+(1)$. In order to show $\mathcal{Q}^+(v_0, v_1) \supseteq \mathcal{Q}^+(1)$, consider an arbitrary element $\mathcal{O}_b(g, h) \in \mathcal{Q}^+(1)$. From $\mathcal{O}_b(g, h) \notin \mathcal{Q}(b(v_0))$ and $\mathcal{O}_b(g, h) \in \mathcal{Q}(t(v_0))$ it follows that $g \ge \xi(v_0^b)$. Furthermore, $g = \xi(v_0^b)$ and $h = \psi(v_1^b)$ are contradictory to the max. property of v_0^b and the min. property of v_1^b, respectively (see obstacle b_1 and b_2 in Fig. 6). At the same time $\xi(v_0^b) < g < \xi(v_1^b)$ and $\psi(v_0^b) < h < \psi(v_1^b)$ are in contradiction to the minimality of v_1^b in $\mathcal{D}(v_0^b)$ (obstacle b_3 in Fig. 6), from which we conclude $g \ge \xi(v_1^b)$ and $h \le \psi(v_1^b)$, and thus $\mathcal{Q}^+(1) \subseteq \mathcal{Q}^+(v_0, v_1)$.

Now assume (12) is true for some k with $1 \le k < n$. Then the path obtained by appending edge (v_k, v_{k+1}) has length

$$q_k w_{\text{ext}} - \sum_{o \in \mathcal{Q}^-(k)} \lambda(o) + \sum_{o \in \mathcal{Q}^+(k)} \lambda(o) + l(v_k, v_{k+1}) \tag{13}$$

$$= q_{k+1} w_{\text{ext}} - \sum_{o \in \mathcal{Q}^-(k) \uplus \mathcal{Q}^-(v_k, v_{k+1})} \lambda(o) + \sum_{o \in \mathcal{Q}^+(k) \uplus \mathcal{Q}^+(v_k, v_{k+1})} \lambda(o) \tag{14}$$

where $q_i = ||t(v_i) - t(v_0)||_1$. Now assume $(v_k, v_{k+1}) \in \mathcal{E}_b$ (for $(v_k, v_{k+1}) \in \mathcal{E}_r$ a symmetric argument applies). Then it is easy to see, that

$$\mathcal{Q}^-(k+1) = \mathcal{Q}^-(k) \uplus \mathcal{Q}^-(v_k, v_{k+1}) \quad \text{and} \tag{15}$$

$$\mathcal{Q}^+(k+1) = \mathcal{Q}^+(k) \uplus \mathcal{Q}^+(v_k, v_{k+1}) \tag{16}$$

if $(v_{i-1}, v_i) \in \mathcal{E}_b,\ \forall 1 \leq i \leq k$ (Fig. 7a), and

$$\mathcal{Q}^-(k+1) = \{\mathcal{Q}^-(k) \uplus \mathcal{Q}^-(v_k, v_{k+1})\} \setminus \mathcal{Q}^+(v_k, v_{k+1}) \quad \text{and} \tag{17}$$

$$\mathcal{Q}^+(k+1) = \mathcal{Q}^+(k) \tag{18}$$

otherwise (Fig. 7b). In both cases (12) follows by induction.

Now let again $q_n = ||t(v_n) - t(v_0)||_1$ and $\hat{q}_n = ||t(v_n) - b(v_0)||_1$. Then by extending path $\hat{p}$ by an arc from an appropriate base node $\mathcal{S}(b(v_0))$, $\mathcal{S} \in \{D, H, V, B\}$, to bpg node v_0 we obtain a path p of desired length

$$\hat{q}_n w_{\text{ext}} + r \cdot w_{\text{open}} - \sum_{o \in \{\mathcal{Q}(t(v_n)) \setminus \mathcal{Q}(b(v_0))\}} \lambda(o), \tag{19}$$

where the number r of gaps we are opening in cell $b(v_0)$ is determined by the type $\mathcal{S}$ of the base node in which path p originates. More precisely, $r = 2$ if $\mathcal{S} = D$, $r = 1$ if $\mathcal{S} \in \{H, V\}$ and $r = 0$ if $\mathcal{S} = B$.

Note that the termination condition of sequence $\langle v_i \rangle_0^n$ implies $\forall o \in \{\mathcal{Q}(t(v_n)) \setminus \mathcal{Q}(b(v_0))\} : o \in \mathcal{Q}((l', m'))$ and therefore the claim of the lemma follows. □

Lemma 4.5 *Given strings s and t of length n_s and n_t, respectively, consider arbitrary but fixed indices $1 \leq l < l' < n_s$ and $1 \leq m < m' < n_t$. There is a path of length $w_{\text{ext}} \cdot ||(l', m') - (l, m)||_1 + 2 \cdot w_{\text{open}} - \sum_{o \in \{\mathcal{Q}((l',m')) \setminus \mathcal{Q}((l,m))\}} \lambda(o)$ from $D(l, m)$ to $B(l', m')$.*

Proof Starting from node $D(l, m)$, traversing exclusively gap arcs, we enter the bpg from a node in cell (l_b, m_b), from which we can not proceed without entering a forbidden obstacle. Cell (l_b, m_b) must be the base of a pair of conflicting obstacles (see Fig. 3a). We thus construct a sequence $\langle v_i \rangle_0^n$ of pairs of conflicting obstacles as described in the proof of Lemma 4.4 to determine the path through the bpg. If we now can find a path from the B-node in cell $t(v_n)$ to node $B(l', m')$ using exclusively gap arcs that are not entering any forbidden obstacles the overall path from $D(l, m)$ to $B(l', m')$ has desired length $w_{\text{ext}} \cdot ||(l', m') - (l, m)||_1 + 2 \cdot w_{\text{open}} - \sum_{o \in \mathcal{Q}((l',m')) \setminus \mathcal{Q}((l,m))} \lambda(o)$. Otherwise we reach again the base of a pair of forbidden conflicting obstacles and we apply Lemma 4.4 again to jump over forbidden obstacles. □

Finally we show, that we do not overestimate the optimal path length.

Lemma 4.6 *Given strings s and t of length n_s and n_t, respectively, consider arbitrary but fixed indices $1 \leq l < l' < n_s$ and $1 \leq m < m' < n_t$. Any path from $D(l, m)$ to $B(l', m')$ that uses only gap arcs has length at most $w_{\text{ext}} \cdot ||(l', m') - (l, m)||_1 + 2 \cdot w_{\text{open}} - \sum_{o \in \{\mathcal{Q}((l',m')) \setminus \mathcal{Q}((l,m))\}} \lambda(o)$.*

Proof For the sake of simplicity consider an arbitrary path that enters the bpg only once from a node $\mathcal{S}(l_b, m_b)$ and returns to the original dynamic programming graph at a node $B(t_n)$. Then obstacles in $\mathcal{Q}((l', m')) \setminus \mathcal{Q}((l, m))$ can be subdivided into three disjoint groups. Obstacles that enclose $(l_\mathfrak{b}, m_\mathfrak{b})$, obstacles in $\mathcal{Q}(t_n) \setminus \mathcal{Q}((l_\mathfrak{b}, m_\mathfrak{b}))$ and obstacles that do not enclose t_n. Obstacles from first and third group must be entered and thus paid by the sequence of gap arcs leading from $D(l,m)$ to $\mathcal{S}(l_\mathfrak{b}, m_\mathfrak{b})$ (formulas (5)–(7)), and from $B(t_n)$ to $B(l', m')$ (formula (7)), respectively. The length of an arbitrary path p'–from $\mathcal{S}(l_\mathfrak{b}, m_\mathfrak{b})$ to $B(t_n)$ through the bpg differs from path p induced by sequence $\langle v_i \rangle_0^n$ and constructed in Lemma 4.4 only in two aspects. First, for an edge (v_k, v_{k+1}) on path $\hat{p}$ we have to relax (16) to $\mathcal{Q}^+(k+1) \supseteq \mathcal{Q}^+(k) \uplus \mathcal{Q}^+(v_k, v_{k+1})$ and (17) to $\mathcal{Q}^-(k+1) \subseteq \mathcal{Q}^-(k) \setminus \mathcal{Q}^+(v_k, v_{k+1})$. Intuitively, when traversing edge $(v_k, v_{k+1}) \in \mathcal{E}_b$ in Fig. 7a, the shaded rectangle may still contain obstacles. And second, $\{\mathcal{Q}(t_n) \setminus \mathcal{Q}((l_\mathfrak{b}, m_\mathfrak{b}))\} \subseteq \mathcal{Q}((l', m'))$ (see termination condition of sequence $\langle v_i \rangle_0^n$) does not necessarily hold. As a consequence, obstacles that contribute to the penalty of path p also contribute to the penalty of $\hat{p}$ and the claim follows. □

5 Improving the Lagrangian relaxation bound

Recall that (LR_λ) is the problem of computing all extended pairwise alignments for a given set of multipliers λ and $v(LR_\lambda)$ is its objective function value. Moreover, (P) is the multiple sequence alignment problem itself.

Since the optimal value $v(LR_\lambda)$ is an upper bound on the optimal value of (P) for all multiplier vectors $\lambda \in \mathbb{R}_+^m$, $m = |\mathcal{M}|$, we are interested in solving the problem

$$(LR) \quad \min_{\lambda \geq 0} v(LR_\lambda)$$

to obtain tighter bounds for our branch-and-bound algorithm.

5.1 Subgradient optimization

It is well known that the *Lagrangian function* $f(\lambda) = v(LR_\lambda)$ (for our case where (P) is a maximization problem) is a convex function of λ, but it is not differentiable at points, where the optimal solution of (LR_λ) is not unique. A commonly used approach to determine near-optimal Lagrangian multipliers efficiently is based on the vector of *subgradients* $g(\lambda) \in \mathbb{R}^m$, associated with a given λ. The set $\partial f(\lambda^0)$ of all subgradients of $f(\lambda)$ at a point λ^0 is always nonempty, and one can show that the vector

$$g_M(\lambda^0) = r - 1 - \sum_{j=1}^{r} \bar{z}_{(u_j, u_{j+1})}, \qquad M \in \mathcal{M} \tag{20}$$

is contained in $\partial f(\lambda^0)$, where $\bar{z}$ is an optimal solution to (LR_{λ^0}). The iterative approach proposed by Held and Karp (1971) generates a sequence $\lambda^0, \lambda^1, \ldots$ of Lagrangian multipliers by taking at iteration k a step along a subgradient of $f(\lambda^k)$, projecting the resulting point onto the nonnegative orthant:

$$\lambda_M^{k+1} = \max\left\{0, \lambda_M^k + \theta \frac{v(LR_{\lambda^k}) - LB}{\sum_{M' \in \mathcal{M}} g_{M'}^2} g_M(\lambda^k)\right\}, \qquad M \in \mathcal{M} \tag{21}$$

where LB is a lower bound on $v(P)$, and θ is a step size parameter assuming values in $(0, 2]$. As to the adaption of scalar step size θ, our approach differs from the classical Held-Karp method, which halves parameter θ when there is no upper bound improvement for a certain number of consecutive iterations. If the best and worst upper bounds computed in the last p iterations differ by more than 1%, we suspect that we are "overshooting" and thus we halve the current value of θ. If, in contrast, the two values are within 0.1% from each other, we overestimate $v(LR_{\lambda^*})$, where λ^* is an optimal solution to (LR), and therefore increase θ by a factor of 1.5. Similarly to Caprara et al. (1999), we experienced a faster convergence to near optimal multipliers using this strategy, compared to the classical approach.

As (2) involves exponentially many mixed cycle inequalities that would have to be dualized, formula (21) can not be applied in a straightforward way, but we use the relax-and-cut framework outlined below.

5.2 Relax-and-cut

In the traditional case of the subgradient method (SM), when the number of dualized constraints is not too large, Beasley (1993) reported good practical convergence to $v(LR)$, when setting $g_i = 0$ whenever $g_i \geq 0$ and $\lambda_i = 0$, for $i \in 1, \dots, m$, i.e. if an inequality whose multiplier is 0 is not violated. We extend this idea by setting $g_M = 0$ for all M with $\lambda_M = 0$ whose corresponding mixed cycle inequalities are not violated by the Lagrangian solution. These multipliers would remain zero valued at the end of the current iteration and thus would not directly contribute to $v(LR_\lambda)$, at any given SM iteration. We call the corresponding constraints *inactive inequalities*. Conversely, we call inequalities, whose associated multiplier may directly contribute to the Lagrangian objective function, *active inequalities*. These are the constraints (2) that are violated by the Lagrangian solution and those inequalities that have nonzero multipliers associated with them. Otherwise the value $\sum_{M \in \mathcal{M}} g_M$ would be very high, resulting in virtually unchanged multipliers from iteration to iteration. We therefore apply (21) exclusively to active inequalities, as suggested in Lucena (1993).

This dynamic scheme, where the pool of active inequalities may continuously change, heavily relies on the ability to identify inequalities that are violated by the Lagrangian solution. In order to prevent the set of active inequalities from growing too rapidly we restrict the separation problem to mixed cycle inequalities, that are most violated by the average of the last h solutions. Experiments show, that this modification improves the rate of convergence dramatically (Table 5).

6 Experiments

We have implemented our Lagrangian approach in C++ using the LEDA-library (Mehlhorn and Näher 1999) and have embedded it into a branch-and-bound framework. The lower bounds in each bb node are computed by selecting, in a greedy fashion, edges from the set $\{e \in E_A \mid \bar{x}_e = 1\}$ that satisfy conditions (1)–(3). The weights for the alignment edges were obtained by the BLOSUM62 amino acid substitution matrix, whereas the gap arcs were assigned a weight that was computed as $4l + 6$, where l is the number of characters in the corresponding gap.

We tested our implementation on a set of instances of the BAliBASE library, including reference alignments from last inception, version 3. The benchmark alignments from reference 1 (R1) contain 4 to 6 sequences and are subdivided into three groups of different length (short, medium, long). They are further categorized into three subgroups by the degree of similarity between the sequences (group V1: identity $< 25\%$, group V2: identity 20–40%, group V3: identity $> 35\%$). In BAliBASE 3 two datasets are prepared. The homologous region set, which is similar to BAliBASE 2, and the full length set, where sequences contain non-homologous residues.

The purpose of the experiments is twofold. First, we want to evaluate the complexity of instances our current implementation is able to solve in reasonable time. Second, we want to assess the quality of alignments that are *optimal* in our model of the multiple sequence alignment problem. For the latter we used an evaluation program provided by BAliBASE that computes a score between 0 and 1, indicating the degree of accordance with the hand-made reference alignment, which uses structural information (see Table 4).

We compared our implementation, which we will call LASA (LAgrangian Sequence Alignment) with MSA (Lipman et al. 1989), COSA (Althaus et al. 2006), T-COFFEE (Notredame et al. 2000), CLUSTALW (Thompson et al. 1994), MAFFT (Katoh et al. 2005) and MUSCLE (Edgar 2004). The multiple sequence alignment program MSA is based on dynamic programming and uses the so called quasi-affine gap cost model, a simplification of the (natural) affine gap cost model. The branch-and-cut algorithm COSA is based on the same ILP formulation and uses CPLEX as LP-solver.

We ran the experiments on a system with a 2,39 GHz AMD Opteron Processor with 8 GB of RAM. Any run that exceeded a CPU time limit of 12 hours was considered unsuccessful. Tables 1, 2 and 3 report our results on short, medium sized and long instances from reference 1, respectively. The columns have the following meaning:

- Instance: Name of the instance, along with an indication (k, n) of the number of sequences and the overall number of characters;
- Heur: Value of the initial feasible solution found by COSA or MSA;
- PUB: Pairwise upper bound;
- Root: Value of the Lagrangian upper bound at the root node of the branch-and-bound tree;
- Opt: Optimal solution value;
- #Nodes: Number of branch-and-bound subproblems solved;
- #Iter: Total number of iterations during the subgradient optimization;
- Time: Total running time in seconds.

Although MSA reduces the complexity of the problem by incorporating quasi-affine gap costs into the multiple alignment, it could hardly solve instances with a moderate degree of similarity. In contrast, our preliminary implementation outperforms the CPLEX based approach COSA, the only method known till now to solve the MSA problem exactly. COSA was not able to solve any of the medium sized or long benchmark alignments, while LASA found the optimal solution within minutes. This is mainly because the LPs are quite complicated to solve. Moreover, one instance crashed as an LP could not be solved by CPLEX (Table 1)

Table 1 Results on short instances from reference 1

Instance	Heur	PUB	Root	Opt	LASA			COSA	MSA
					#Nodes	#Iter	Time	Time	Time
Reference 1 Short, V3									
1aho (5/320)	877	987	884	881	7	1,089	<1	1:29	–
1csp (5/339)	1,457	1,473	1,457	1,457	1	17	<1	1	<1
1dox (4/374)	749	782	751	750	3	253	3	30	<1
1fkj (5/517)	1,578	1,675	1,585	1,578	3	348	13	6:04	–
1fmb (4/400)	1,333	1,353	1,333	1,333	1	13	<1	2	<1
1krn (5/390)	1,523	1,558	1,523	1,523	1	104	1	6	6
1plc (5/470)	1,736	1,824	1,736	1,736	1	218	6	4:24	20:14
2fxb (5/287)	1,341	1,352	1,341	1,341	1	11	<	< 1	<1
2mhr (5/572)	2,364	2,406	2,364	2,364	1	65	3	2	17
9rnt (5/499)	2,550	2,573	2,550	2,550	1	39	1	4	<1
Reference 1 Short, V2									
1aab (4/291)	231	257	231	231	1	100	<1	4	< 1
1csy (5/510)	649	769	649	649	1	393	17	3:01	–
1fjlA (6/398)	674	731	676	674	5	561	12	34	–
1hfh (5/606)	903	1,067	911	903	3	411	33	–	–
1hpi (4/293)	386	439	386	386	1	298	4	53	7
1pfc (5/560)	994	1,139	1,004	994	11	1,387	1:48	37:46	–
1tgxA (4/239)	247	317	247	247	1	566	9	53	–
1ycc (4/426)	117	309	202	200	7	1,865	2:19	–	–
3cyr (4/414)	515	615	522	515	7	983	38	–*	45
Reference 1 Short, V1									
1aboA (5/297)	−685	−476	−604	−676	3,497	417,260	11:04:02	–	–
1tvxA (4/242)	−409	−260	−358	−405	777	122,785	1:59:44	–	–
1idy (5/269)	−420	−273	−356	−414	4,193	678,592	12:00:48	–	–
1r69 (4/277)	−326	−207	−289	−326	253	54,668	58:40	–	–
1ubi (4/327)	−372	−246	−330	−372	215	43,620	1:12:57	–	–
1wit (5/484)	−198	−25	−186	−197	15	4,221	7:42	–	–
2trx (4/362)	−182	−88	−178	−182	5	2,186	3:04	–	–

*With the COSA-code, the instance 3cyr crashed after about one hour of computation time as the LP-solver was not able to solve the underlying LP

Concerning the quality of the alignments, our approach ranks among the best programs implemented so far (see Table 4). The quality could be probably improved by a more careful choice of the objective function. In our current implementation we use a fixed objective for all instances, no matter what their level of identity is.

Furthermore, we give computational evidence for the effectiveness of our novel approach to select violated inequalities to be added to our constraint pool. Consider-

Table 2 Results on medium sized instances from reference 1. COSA and MSA were not able to solve any of these benchmark alignments. Results on group V1 are omitted, since LASA was not able to solve these instances in the allowed time frame

Instance	Heur	PUB	Root	Opt	#Nodes	#Iter	Time
Reference 1 Medium, V3							
1amk (5/1241)	5,668	5,728	5,669	5,669	1	60	8
1ar5A (4/794)	2,303	2,357	2,304	2,303	3	262	20
1ezm (5/1515)	8,378	8,466	8,378	8,378	1	105	23
1led (4/947)	2,150	2,282	2,158	2,150	33	1,435	3:54
1ppn (5/1083)	4,718	4,811	4,729	4,724	23	925	3:10
1pysA (4/1005)	2,730	2,796	2,732	2,730	3	223	28
1thm (4/1097)	3,466	3,516	3,468	3,468	3	233	30
1tis (5/1413)	5,854	5,999	5,874	5,856	83	2,993	18:31
1zin (4/852)	2,357	2,411	2,361	2,357	13	625	1:03
5ptp (5/1162)	4,190	4,329	4,233	4,205	193	8,337	35:48
Reference 1 Medium, V2							
1ad2 (4/828)	1,195	1,270	1,197	1,195	7	419	42
1aym3 (4/932)	1,544	1,664	1,551	1,544	17	1,060	2:37
1gdoA (4/988)	980	1,201	1,003	984	459	31,291	2:38:36
1ldg (4/1240)	1,526	1,640	1,539	1,526	41	2,160	8:32
1mrj (4/1025)	1,461	1,608	1,473	1,464	27	1,681	5:29
1pgtA (4/828)	683	808	691	690	9	926	2:05
1pii (4/1006)	1,099	1,256	1,103	1,100	23	1,320	4:54
1ton (5/1173)	1,550	1,898	1,609	1,554	807	44,148	5:32:47

Table 3 Results on long sequences from reference 1. Only three instances could be solved by LASA. MSA and COSA were not able to solve any of these benchmark alignments. Instance 3pmg was solved once with an initial lower bound obtained by MSA (7363) and once with the optimal value (7418) computed by LASA itself

Instance	Heur	PUB	Root	Opt	#Nodes	#Iter	Time
1ad3 (4/1746)	5,355	5,424		5,358	21	734	4:25
actin (5/1924)	8,018	8,178	8,039	8,022	45	2,138	19:41
3pmg (4/2224)	7,363	7,602	7,460	7,418	1,397	53,350	12:50:50
3pmg (4/2224)	7,418	7,602	7,448	7,418	119	4,789	1:08:37

ing the average of the last h solutions of the Lagrangian relaxation instead of looking only at the current solution ($h = 1$) dramatically reduces the number of iterations (see Table 5). Only short sequences of high identity (short, V3) could be solved for $h = 1$. Moreover, this table shows that the extended pairwise alignment problems are solved at least twice as fast when using the A^* approach.

The columns in Table 5 have the following meaning:

Table 4 Average score of the alignments computed by different programs. Only instances that have been solved by LASA were considered. In BAliBASE 3.0 full length sequences (full) and instances from the homologous region set (hom) are distinguished

Group	LASA	T-COFFEE	CLUSTALW	MAFFT	MUSCLE
BAliBASE 2.0					
Short V1	0.969	0.968	0.984	0.988	0.970
Short V2	0.865	0.819	0.936	0.948	0.811
Short V3	0.512	0.340	0.562	0.882	0.537
Medium V1	0.944	0.952	0.943	0.969	0.969
Medium V2	0.933	0.886	0.911	0.901	0.895
Long V1	0.960	0.941	0.933	0.976	0.982
BAliBASE 3.0					
RV11 full	0.942	0.966	0.935	0.939	0.952
RV11 hom	0.795	0.672	0.764	0.819	0.780
RV12 full	0.918	0.900	0.918	0.919	0.905
RV12 hom	0.894	0.876	0.895	0.882	0.888

Table 5 We give the number of iterations needed by our approach for different numbers h of solutions that were considered to compute the average Lagrangian solution. The default is $h = 10$. The last column gives the time spent in the root node if we resign to use the A^* approach

Instance	$h=1$	$h=2$	$h=20$	$h=30$	LASA (A*, $h=10$)		DynProg, $h=10$
	#Iter	#Iter	#Iter	#Iter	#Iter	Time	Time
1aho (5/320)	748,470	2,496	1,194	1,283	1,089	10	22
1csp (5/339)	17	14	19	19	17	<1	<1
1dox (4/374)	80,001	271	211	207	253	1	5
1fkj (5/517)	316,072	849	707	676	348	9	25
1fmb (4/400)	1,372	14	14	14	13	<1	<1
1krn (5/390)	191,281	634	148	155	104	1	8
1plc (5/470)	232,591	489	642	513	218	06	14
2fxb (5/287)	16,425	15	11	11	11	<1	<1
2mhr (5/572)	60,005	93	116	177	65	3	8
9rnt (5/499)	54	49	40	40	39	1	3

- Instance: Name of the instance, along with an indication (k, n) of the number of sequences and the overall number of characters;
- $h = \cdot$: The number of solutions that were considered to compute an average Lagrangian solution;
- LASA: Default version of LASA, i.e. $h = 10$ and using the A* approach;
- DynProg: LASA without using the A* approach;
- #Iter: Number of iterations needed by a specific version of LASA;
- Time: Total running time in seconds needed by a specific version of LASA;

Table 6 For the first four benchmark alignments of each subgroup of short and medium sized instances we compare the size of the underlying graphs of the simple respectively improved algorithm (with and without "transitive reduction"). The figures show a snap-shot after the last iteration in the root node of the branch and bound tree of the average number of blue and red obstacles between each pair of sequences (#Obst), the average number of nodes in the bypass graph (# BPG-Nodes), the average number of arcs of the bpg (#BPG-Arcs), including arcs connecting the bpg to the dynamic programming graph, and the average number of additional arcs needed when no bpg is used (#Arcs)

Instance	#Obst	#BPG-Nodes	#BPG-Arcs	#Arcs
Reference 1 Short, V3				
1aho (5/320)	32	7	42	205
1csp (5/339)	6	0	0	0
1dox (4/374)	18	3	18	56
1fkj (5/517)	29	6	35	137
Reference 1 Short, V2				
1aab (4/291)	18	3	18	51
1csy (5/510)	57	11	58	333
1fjlA (6/398)	13	2	13	33
1hfh (5/606)	61	17	91	760
Reference 1 Short, V1				
1aboA (5/297)	63	40	236	4047
1tvxA (4/242)	82	61	373	8979
1idy (5/269)	51	21	120	1314
1r69 (4/277)	74	29	165	1934
Reference 1 Medium, V3				
1amk (5/1241)	17	1	7	9
1ar5A (4/794)	39	10	56	258
1ezm (5/1515)	24	5	26	83
1led (4/947)	65	13	72	468
Reference 1 Medium, V2				
1ad2 (4/828)	51	11	61	360
1aym3 (4/932)	54	21	122	1286
1gdoA (4/988)	104	22	121	1168
1ldg (4/1240)	67	14	76	491

Finally, we try to assess the practical performance of the simple algorithm and both versions of the improved algorithm (with and without "transitive reduction") by considering the size of the underlying graph structure after the last iteration in the root node of the branch and bound tree. Table 6 indicates that for a bpg $G = (\mathcal{V}, \mathcal{E})$, $\mathcal{O}(|\mathcal{O}|^2)$ and $\mathcal{O}(|\mathcal{O}|^3)$ are rather pessimistic estimates for $|\mathcal{V}|$, respectively $|\mathcal{E}|$, and therefore we expect the running time of the simple algorithm ($\mathcal{O}(n_s^2 n_t^2 |\mathcal{O}|)$) to be significantly larger than the running time of the improved algorithm using the bpg ($\mathcal{O}(nm + |\mathcal{V}| + |\mathcal{E}|)$). Moreover, the "transitive reduction" obtained by introducing the bypass graph reduces the number of additional arcs considerably.

7 Conclusion

We have constructed a Lagrangian relaxation of the multiple sequence alignment ILP formulation that allowed us to obtain strong bounds by solving a generalization of the pairwise alignment problem. By utilizing these bounds in a branch-and-bound manner we achieved running times that outperform all other exact or almost exact methods. We plan to integrate our implementation into the software project SEQAN currently developed by the Free University of Berlin.

Besides optimizing our implementation for speed an important issue in our future work will be to extend the scheme to Volume and to Bundle algorithms. A more sophisticated Lagrangian heuristic for computing lower bounds in the bb nodes will be necessary to be able to solve instances of larger size (compare instance 3pmg in Table 3).

References

Althaus E, Caprara A, Lenhof H-P, Reinert K (2002) Multiple sequence alignment with arbitrary gap costs: Computing an optimal solution using polyhedral combinatorics. In: Lengauer T, Lenhof H-P (eds) Proceedings of the European conference on computational biology, Saarbrücken, October 2002. Bioinformatics, vol 18. Oxford University Press, London, pp S4–S16

Althaus E, Caprara A, Lenhof H-P, Reinert K (2006) A branch-and-cut algorithm for multiple sequence alignment. Math Program 105:387–425

Beasley J (1993) Lagrangian relaxation. In: Modern heuristic techniques for combinatorial problems. Blackwell Scientific, Oxford

Caprara A, Fischetti M, Toth P (1999) A heuristic method for the set cover problem. Oper Res 47:730–743

Carrillo H, Lipman DJ (1988) The multiple sequence alignment problem in biology. SIAM J Appl Math 48(5):1073–1082

Delcher A, Kasif S, Fleischmann R, Peterson J, White O, Salzberg S (1999) Alignment of whole genomes. Nucleic Acids Res 27:2369–2376

Edgar RC (2004) Muscle: multiple sequence alignment with high accuracy and high throughput. Nucleic Acids Res 32(5):1792–1797

Elias I (2003) Settling the intractability of multiple alignment. In: Proc. of the 14th ann. int. symp. on algorithms and computation (ISAAC'03). Lecture notes in computer science, vol 2906. Springer, Berlin, pp 352–363

Eppstein D (1990) Sequence comparison with mixed convex and concave costs. J Algorithms 11:85–101

Fisher M (1994) Optimal solutions of vehicle routing problems using minimum k-trees. Oper Res 42:626–642

Garey M, Johnson D (1979) Computers and intractability: a guide to the theory of NP-completeness. Freeman, New York

Gupta S, Kececioglu J, Schaeffer A (1995) Improving the practical space and time efficiency of the shortest-paths approach to sum-of-pairs multiple sequence alignment. J Comput Biol 2:459–472

Gusfield D (1997) Algorithms on strings, trees and sequences: computer science and computational biology. Cambridge University Press, Cambridge

Held M, Karp R (1971) The traveling salesman problem and minimum spanning trees: part II. Math Program 1:6–25

Katoh K, Kuma K, Toh H, Miyata T (2005) MAFFT version 5: improvement in accuracy of multiple sequence alignment. Nucleic Acids 33:511

Larmore L, Schieber B (1990) Online dynamic programming with applications to the prediction of RNA secondary structure. In: Proceedings of the first symposium on discrete algorithms, pp 503–512

Lermen M, Reinert K (2000) The practical use of the $\mathcal{A}^*$ algorithm for exact multiple sequence alignment. J Comput Biol 7(5):655–673

Lipman D, Altschul S, Kececioglu J (1989) A tool for multiple sequence alignment. Proc Nat Acad Sci US Am 86:4412–4415

Lucena A (1993) Steiner problem in graphs: Lagrangean relaxation and cutting-planes. COAL Bull 21:2–7

Mehlhorn K, Näher S (1999) The LEDA platform of combinatorial and geometric computing. Cambridge University Press, Cambridge. See also http://www.mpi-sb.mpg.de/LEDA/

Notredame C, Higgins DG, Heringa J (2000) T-Coffee: a novel method for fast and accurate multiple sequence alignment. J Mol Biol 302(1):205–217

Reinert K (1999) A polyhedral approach to sequence alignment problems. PhD thesis, Universität des Saarlandes, 1999

Reinert K, Lenhof H-P, Mutzel P, Mehlhorn K, Kececioglu J (1997) A branch-and-cut algorithm for multiple sequence alignment. In: Proceedings of the first annual international conference on computational molecular biology (RECOMB-97), pp 241–249

Reinert K, Stoye J, Will T (2000) An iterative method for faster sum-of-pairs multiple sequence alignment. Bioinformatics 16(9):808–814

Sankoff D, Kruskal JB (1983) Time warps, string edits and macromolecules: the theory and practice of sequence comparison. Addison–Wesley, Reading

Subramanian AR, Weyer-Menkhoff J, Kaufmann M, Morgenstern B (2005) DIALIGN-T: An improved algorithm for segment-based multiple sequence alignment. BMC Bioinformatics 6:66

Thompson JD, Higgins DG, Gibson TJ (1994) CLUSTAL W: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice. Nucleic Acids Res 22(22):4673–4680