A Lagrangian relaxation approach for the multiple sequence alignment problem *

Ernst Althaus[1] and Stefan Canzar[2]

[1] Max-Planck Institut für Informatik, Stuhlsatzenhausweg 85, D-66123 Saarbrücken, Germany
[2] Université Henri Poincaré, LORIA, B.P. 239, 54506 Vandœvre-lès-Nancy, France

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.

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 $\mathcal{S} = \{s^1, s^2, \ldots, s^k\}$ 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 $\mid s \mid$ denote the number of characters in the string and s_l the lth character of s, for $l = 1, \ldots, \mid s \mid$. We will assume that $\mid s^i \mid \geq 4$ for all strings s^i and let $n := \sum_{i=1}^{k} \mid s^i \mid$.

An *alignment* $\mathcal{A}$ of $\mathcal{S}$ is a set $\bar{\mathcal{S}} = \{\bar{s}^1, \bar{s}^2, \cdots, \bar{s}^k\}$ 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.

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.

* Supported by the German Academic Exchange Service (DAAD)

Moreover, a *gap* in s^i with respect to s^j is a maximal sequence $s^i_l\ s^i_{l+1}\ \ldots\ s^i_m$ 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 [12]. 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 $\mathcal{A}$ [6].

If the number k of sequences is fixed the multiple alignment problem for sequences of length n can be solved in time and space $\mathcal{O}\left(n^k\right)$ with (quasi)-affine gap costs [11, 15, 19, 20]. More complex gap cost functions add a polylog factor to this complexity [8, 14]. However, if the number k of sequences is not fixed, Wang and Jiang [22] proved that multiple alignment with SP score is $\mathcal{NP}$-complete by a reduction from *shortest common supersequence* [10]. 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., [4, 7]).

In [3, 2] Althaus et al. 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 Section 2 we review the ILP formulation of the multiple sequence alignment problem, whose Lagrangian relaxation is described in section 3. Our algorithm for solving the resulting problem is introduced in section 4. Finally, computational experiments on a set of real-world instances are reported in section 5.

2 Previous Work

In [3] Althaus et al. use a formulation for the multiple sequence alignment problem as an ILP given by Reinert in [18].

For ease of notation, they define the *gapped trace graph*, a mixed graph whose node set corresponds to the characters of the strings and whose edge set is partitioned in 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 $V^i = \{u^i_j \mid 1 \le j \le |s^i|\}$, $E_A = \{uv \mid u \in V^i, v \in V^j, i \ne j\}$ and $A_P = \{(u^i_l, u^i_{l+1}) \mid 1 \le i \le k \text{ and } 1 \le l < |s^i|\}$ (see figure 1). Furthermore, we denote with $\mathcal{G} = \{(u, v, j) \mid u, v \in V^i, j \ne i\}$ the set of all possible gaps.

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 [18] showed that solutions to the alignment problem are the $\{0,1\}$-assignments to the variables such that

1. we have pairwise alignments between every pair of strings,
2. there are no mixed cycles, i.e. in the subgraph of the gapped trace graph consisting of the positioning arcs A_P and the 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 figure 1),
3. transitivity is preserved, i.e. if u is aligned with v and v with w then u is aligned with w, for $u, v, w \in V$.

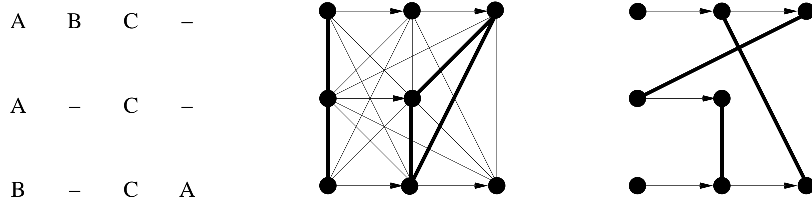

Fig. 1. The graph in the middle is the gapped trace graph for the alignment problem given in the left part. The thick edges specify the alignment given in the left part. The alignment edges in the right part can not be realized at the same time in an alignment. Together with appropriate arcs of A_P, they form a mixed cycle.

These three conditions are easily formulated as linear constraints. 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 the optimal alignment (whose overall weight is maximized) satisfying conditions (1)-(3) 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 we use additional variables $z_{(u,v)}$ for $u \in V^i, v \in V^j, i \ne j$, with the property that $z_{(u,v)} = 1$ iff at least one character of the string of u lying behind u is aligned to a character of the string of v lying before v, i.e. $z_{(u^i_l, v^j_m)} = 1$, iff there is $l' \ge l$ and $m' \le m$ with $x_{v^i_{l'} v^j_{m'}} = 1$. This condition is captured by the inequalities

$$0 \le z \le 1, \quad z_{(u^i_l, v^j_m)} \ge z_{(u^i_{l+1}, v^j_m)} + x_{u^i_l, v^j_m} \text{ and } z_{(u^i_l, v^j_m)} \ge z_{(u^i_l, v^j_{m-1})} + x_{u^i_l, v^j_m}. \tag{4}$$

In the following, we describe the inequalities used in [3] to enforce (2). We resign to explicitly specify the inequalities enforcing (1) and (3), as they are not crucial for the understanding of our approach.

Using these additional variables, we can define facets that guarantee (2) as follows. Let $A_A = \{(u,v) \mid u \in V^i, v \in V^j, i \neq j\}$, i.e. for each undirected edge $uv \in E_A$, we have the two directed arcs (u,v) and (v,u) in A_A. Let $M \subseteq A_A \cup A_P$ be a cycle in $(V, A_A \cup A_P)$ that contains at least one arc of A_P. We call such a cycle a *mixed cycle*. The set of all mixed cycle inequalities is denoted by $\mathcal{M}$. For a mixed cycle $M \in \mathcal{M}$ the inequality

$$\sum_{e \in M \cap A_A} z_e \leq |M \cap A_A| - 1 \tag{5}$$

is valid and defines a facet under appropriate technical conditions. In particular, there is exactly one arc of A_P in M. These inequalities are called *lifted mixed cycle inequalities*. The constraints 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 $Z^{i,j}$ variables. The constraints (2) and (3) are used to guarantee that all pairwise alignments together form a multiple sequence alignment. We call an assignment of $\{0,1\}$-values to $(X^{i,j}, Y^{i,j}, Z^{i,j})$ such that $(X^{i,j}, Y^{i,j})$ imposes a pairwise alignment and $Z^{i,j}$ satisfies inequalities (4), 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 (2) and relax conditions (3) (during experiments it turned out that relaxing condition (3) is more efficient in practice as dualizing them). Hence our Lagrangian subproblem is an extended pairwise alignment problem. More precisely, if λ_M is the current multiplier for the mixed cycle inequality of $M \in \mathcal{M}$, we have to solve the Lagrangian relaxation problem

$$\begin{aligned} &\sum_{M \in \mathcal{M}} \lambda_M (|M \cap A_A| - 1) \quad + \\ &\quad \max \sum_{e \in E_A} w_e x_e + \sum_{g \in \mathcal{G}} w_g y_g - \sum_{M \in \mathcal{M}} \lambda_M \sum_{e \in M \cap A_A} z_e \qquad (LR_\lambda) \\ &\quad \text{s.t.} (X^{i,j}, Y^{i,j}, Z^{i,j}) \text{ forms an extended pairwise alignment for all } i, j. \end{aligned}$$

We denote its optimal value with $v(LR_\lambda)$. Our approach to obtain tighter bounds efficiently, e.g. to determine near-optimal Lagrangian multipliers to the minimum of the *Lagrangian function* $f(\lambda) = v(LR_\lambda)$, is based on the iterative subgradient method proposed by Held and Karp [13]. Similarly to [5], we experienced a faster convergence if we modify the adaption of scalar step size θ in the subgradient formula in the following way. Instead of simply reducing θ when there is no upper bound improvement for too long, we compare the best and worst upper bounds computed in the last p iterations. If they 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. As the number of inequalities that we dualize is exponential, we modify the subgradient method in a relax-and-cut fashion, as proposed by [9]. Due to lack of space, we resign to give details and refer to [1] for a complete description.

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 of the alignment (s_l, t_m). Then the determination of the optimal alignment value $D(n_s, n_t)$ takes time $\mathcal{O}\left(n_s^2 n_t^2\right)$[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 $\mathcal{O}\left(n_s n_t\right)$.

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 $s_l \in s$, $t_m \in t$, i.e. in all cells (l, m). We call this graph the dynamic programming graph. Each pairwise alignment corresponds to a unique path through this graph, with every arc of the path representing a certain kind of alignment, determined by the type of its target node (figure 2). An *alignment*

[1] The running time can be reduced to $\mathcal{O}\left(n_s^2 n_t\right)$ by distinguishing three different types of alignments [21]

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).

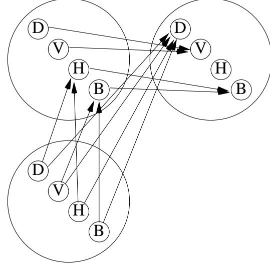

Fig. 2. Three cells of the dynamic programming matrix, with four values (nodes) associated to each of them. Note that arcs (dependencies) are between certain values D, V, H and B, the target node determines the type of the partial alignment.

Now assume some variable $z_{(u,v)}$ is multiplied by a non-zero value in the objective function, as the arc (u, v) is used in at least one mixed cycle inequality, 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 $s = s^i$, $t = s^j$, $u = u_l^i$ and $v = u_m^j$. Then $z_{(u,v)} = 1$, iff there is $l' \geq l$ and $m' \leq m$ such that $x_{u_{l'}^i u_{m'}^j} = 1$ (see definition of variables $z_{(u,v)}$ in (4)). In the dynamic program graph, this corresponds to alignment arcs whose target lies in the lower right rectangle from cell (l, 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 $\mathscr{O}_b(l, m)$ and $\mathscr{O}_r(l, m)$, respectively.

Let the set of all blue and red obstacles be denoted by $\mathscr{O}_b$ and $\mathscr{O}_r$, respectively, and let $\mathscr{O} = \mathscr{O}_b \cup \mathscr{O}_r$. Then the extended pairwise alignment problem is solvable by a dynamic program in $\mathcal{O}\left(n_s^2 n_t^2 |\mathscr{O}|\right)$ time, following the same approach as above: we compute the best alignment of all pairs of prefixes $s_1 \dots s_l$ and $t_1 \dots t_m$ that aligns s_l and t_m, based on on all best alignments of strings $s_1 \dots s_{l'}$ and $t_1 \dots t_{m'}$, for $l' < l$ and $m' < m$. We add the appropriate gap weight to the score of the alignment (s_l, t_m) and subtract all Lagrangian multipliers that are associated with obstacles enclosing (s_l, t_m), but not $(s_{l'}, t_{m'})$.

Definition 1 (Enclosing Obstacles). *The set of enclosing blue obstacles $\mathcal{Q}_b(p)$ of a cell $p = (x, y)$ contains all blue obstacles $\mathscr{O}_b(l, m)$ with $l \leq x, m > y$. Accordingly, $\mathcal{Q}_r(p) = \{\mathscr{O}_r(s, t) \mid s > x, t \leq y\}$. Furthermore we define $\mathcal{Q}(p) = \mathcal{Q}_b(p) \cup \mathcal{Q}_r(p)$.*

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. The key idea is to charge the cost of a Lagrangian multiplier λ as soon as we enter the corresponding obstacle o, i.e. if the target node of the arc is enclosed by o, no matter whether we enter it along an alignment arc or a gap arc. 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. In other words, 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 $\mathscr{O}_b(l,m)$ and a red obstacle $\mathscr{O}_r(l',m')$ *conflicting*, if $l' \geq l$ and $m' \leq m$. The *base* $\mathscr{b}(\mathscr{O}_b(l,m), \mathscr{O}_r(l',m'))$ of a pair of conflicting obstacles is defined as cell $(l-1, m'-1)$, the *target* $\mathscr{t}(\mathscr{O}_b(l,m), \mathscr{O}_r(l',m'))$ 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 $m < m'$. Similarly, a blue (red) obstacle $\mathscr{O}_{b(r)}(l,m)$ *dominates* an obstacle $\mathscr{O}_{b(r)}(l',m')$, iff $(l,m) < (l',m')$. A blue (red) obstacle is *minimal* in $\hat{\mathscr{O}}_b \subseteq \mathscr{O}_b$ ($\hat{\mathscr{O}}_r \subseteq \mathscr{O}_r$), if it is not dominated by any other obstacle in $\hat{\mathscr{O}}_b$ ($\hat{\mathscr{O}}_r$). We denote the set of obstacles that are dominated by a given obstacle o, by $\mathcal{D}(o)$.

It is not difficult to see, that the insertion of arcs from the four nodes of every base $\mathscr{b}$ to the B-node of every target $\mathscr{t}$ such that $\mathscr{b} < \mathscr{t}$, 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 $\mathscr{b}$ to $\mathscr{t}$ plus the penalties implied by obstacles enclosing $\mathscr{t}$, but not $\mathscr{b}$.

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

Definition 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* $\mathcal{E}_b = \{(v,w) \mid w^b \text{ is minimal in } \mathcal{D}(v^b)\}$ *and* $\mathcal{E}_r = \{(v,w) \mid w^r \text{ is minimal in } \mathcal{D}(v^r)\}$*.*

The length l of edges in the bypass graph is chosen appropriately such that there exists a path from any node of base $\mathscr{b}$ to the B-node of every target $\mathscr{t}$ with $\mathscr{b} < \mathscr{t}$, that implies the correct score and such that the length of any such path is upper bounded by that score. We connect the bypass graph to the dynamic programming graph by arcs from all four nodes of every base to all its engendering vertices in $\mathcal{V}$ and by arcs from all $v \in \mathcal{V}$ to the B-node of their

target $\iota(v)$. The length of the former kind of arcs satisfies the same requirements as l, the latter ones are defined to be of length 0.

Concerning the correctness of the dynamic program, we refer to a technical report [1] for details.

4.1 Complexity

Obviously there are at most $|\mathscr{O}|^2$ conflicting pairs of obstacles and hence the number of additional nodes $|\mathcal{V}|$ is at most $|\mathscr{O}|^2$. From definition 2 it follows immediately that the number of additional arcs $|\mathcal{A}|$ is at most $\mathcal{O}\left(|\mathscr{O}|^3\right)$, 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 $\mathcal{O}\left(nm + |\mathscr{O}|^3\right)$.

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 can be at most the scores of the first iteration, i.e. when all multipliers λ are set to 0. Then it is easy to see, that 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.

5 Experiments

We have implemented our Lagrangian approach in C++ using the LEDA-library[17] 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. 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 $> 35\%$).

We compared our implementation, which we will call LASA (LAgrangian Sequence Alignment), with MSA [16] and COSA[3]. 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.

Table 1 reports our results on short and medium sized instances from reference 1. As LASA was able to solve only three of the long instances (and no other program could solve any), we resign to show these results.

The columns in table 1 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;

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.

The running time of LASA and COSA strongly depends on tight initial lower bounds. For example, LASA takes about 13 hours for the long instance 3pmg with the bound obtained by the heuristic and only about one hour with the optimal value used as a lower bound.

Finally, we give computational evidence for the effectiveness of our novel approach to select violated inequalities to be added to our constraint pool. Considering 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 2). Only short sequences of high identity (short, V3) could be solved for $h = 1$. Furthermore, 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 2 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; $h = \cdot$ *:* 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;

6 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.

Instance	Heur	PUB	Root	Opt	LASA #Nodes	LASA #Iter	LASA Time	COSA Time	MSA Time
Reference 1 Short, V3									
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	-
1plc (5/470)	1,736	1,824	1,736	1,736	1	218	6	4:24	20:14
2mhr (5/572)	2,364	2,406	2,364	2,364	1	65	3	2	17
Reference 1 Short, V2									
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	-	-
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 1. Results on instances from reference 1. Results on group medium V1 and two instances of medium V2 are omitted, since no program was able to solve these instances in the allowed time frame. We removed all instances that are solved by LASA within a second. *: With the COSA-code, the instance 3cyr crashed as the LP-solver was not able to solve the underlying LP.

	$h = 1$	$h = 2$	$h = 20$	$h = 30$	LASA (A*, $h = 10$)		DynProg, $h = 10$
Instance	#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	6	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

Table 2. 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.

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.

References

1. E. Althaus and S. Canzar. Solving the extended pairwise alignment problem efficiently. Technical Report MPI-I-2007-1-001, Max-Planck-Institut für Informatik, 66123 Saarbrücken, Germany, May 2007.
2. E. Althaus, A. Caprara, H.-P. Lenhof, and K. Reinert. Multiple sequence alignment with arbitrary gap costs: Computing an optimal solution using polyhedral combinatorics. In T. Lengauer and H.-P. Lenhof, editors, *Proceedings of the European Conference on Computational Biology*, volume 18 of *Bioinformatics*, pages S4–S16, Saarbrücken, October 2002. Oxford University Press.
3. E. Althaus, A. Caprara, H.-P. Lenhof, and K. Reinert. Aligning multiple sequences by cutting planes. *Mathematical Programming*, 105:387–425, 2006.
4. S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman. Basic local alignment search tool. *J. Mol. Biol.*, 215:403–410, 1990.
5. A. Caprara, M. Fischetti, and P. Toth. A heuristic method for the set cover problem. *Operations Research*, 47:730–743, 1999.
6. H. Carrillo and D. J. Lipman. The multiple sequence alignment problem in biology. *SIAM J. Appl. Math.*, 48(5):1073–1082, 1988.
7. A. Delcher, S. Kasif, R. Fleischmann, J. Peterson, W. O., and S. Salzberg. Alignment of whole genomes. *Nucleic Acids Research*, 27:2369–2376, 1999.
8. D. Eppstein. Sequence comparison with mixed convex and concave costs. *Journal of Algorithms*, (11):85–101, 1990.
9. M. Fisher. Optimal solutions of vehcile routing problems using minimum k-trees. *Operations Research*, 42:626–642, 1994.
10. M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W.H. Freeman, 1979.

11. S. Gupta, J. Kececioglu, and A. Schaeffer. 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, 1995.
12. D. Gusfield. *Algorithms on strings, trees and sequences: computer science and computational biology*. Cambridge University Press, Cambridge, 1997.
13. M. Held and R. Karp. The traveling salesman problem and minimum spanning trees: part ii. *Mathematical Programming*, 1:6–25, 1971.
14. L. Larmore and B. Schieber. Online dynamic programming with applications to the prediction of rna secondary structure. In *Proceedings of the First Symposium on Discrete Algorithms*, pages 503–512, 1990.
15. M. Lermen and K. Reinert. The practical use of the $\mathcal{A}^*$ algorithm for exact multiple sequence alignment. *Journal of Computational Biology*, 7(5):655–673, 2000.
16. D. Lipman, S. Altschul, and J. Kececioglu. A tool for multiple sequence alignment. *Proceedings of the National Academy of Sciences of the United States of America*, 86:4412–4415, 1989.
17. K. Mehlhorn and S. Näher. *The LEDA Platform of Combinatorial and Geometric Computing*. Cambridge University Press, Cambridge, 1999. See also `http://www.mpi-sb.mpg.de/LEDA/`.
18. K. Reinert. *A Polyhedral Approach to Sequence Alignment Problems*. PhD thesis, Universität des Saarlandes, 1999.
19. K. Reinert, H.-P. Lenhof, P. Mutzel, K. Mehlhorn, and J. Kececioglu. A branch-and-cut algorithm for multiple sequence alignment. In *Proceedings of the First Annual International Conference on Computational Molecular Biology (RECOMB-97)*, pages 241–249, 1997.
20. K. Reinert, J. Stoye, and T. Will. An iterative methods for faster sum-of-pairs multiple sequence alignment. *BIOINFORMATICS*, 16(9):808–814, 2000.
21. D. Sankoff and J. B. Kruskal. *Time Warps, String Edits and Macromolecules: the Theory and Practice of Sequence Comparison*. Addison Wesley, 1983.
22. L. Wang and T. Jiang. On the complexity of multiple sequence alignment. *J. Comput. Biol.*, 1:337–348, 1994.