# Two Applications of the Divide & Conquer Principle in the Molecular Sciences

G. Brinkman, A.W.M. Dress, S.W. Perrey, J. Stoye

No. 151

March, 1997

# Two Applications of the Divide&Conquer Principle in the Molecular Sciences

G. Brinkman, A.W.M. Dress, S.W. Perrey, J. Stoye

March 13, 1997

## 1 Introduction

One of the most powerful principles for solving complex tasks algorithmically is the so-called *Divide&Conquer Principle.* It has been applied successfully for an amazingly wide range of problems, from combinatorial optimization to matrix multiplication. Its principal idea is to break up a given complex task $T$ appropriately into a reasonable number of less complex tasks $T_1, \ldots, T_k$ so that, by "glueing together" appropriately solutions of those less complex tasks, some or even all solutions of the original complex task $T$ can be found.

Whatever problem the original task were to address, it is often possible to rephrase it as a task to search for certain maps $f \in Y^X$ from a (generally) large set $X$ into a (generally) much smaller set $Y$, that is, for maps $f : X \to Y$ which exhibit a number of very particular, well-specified properties. A Divide&Conquer strategy then can be applied for such a search problem whenever it is possible to break up the set $X$ into subsets $X_1, \ldots, X_k$ (which, of course, may – and in most cases will – be overlapping) and to define specific properties regarding maps $f_i$ from the $X_i$ into $Y$ so that (a) it is comparatively easy to find (some or all) maps $f_i : X_i \to Y$ with the desired properties and (b) it is possible to construct (some or all) maps $f : X \to Y$ from (appropriate) $k$-tuples of maps $(f_1, \ldots, f_k)$ by *concatenation*, that is, by putting

$$f(x) := f_i(x)$$

whenever $x \in X_i$, – provided this is well-defined, that is, provided $x \in X_i \cap X_j$ implies $f_i(x) = f_j(x)$ for all $i, j \in \{1, 2, \ldots, k\}$.

For instance, if we try to find a map $f : X \to Y := \{\pm 1\}$ such that, for some pregiven matrix $C = (c_{ij})_{i,j \in X}$ of real numbers, the value of the quadratic function

$$C(f) := \sum_{i,j \in X} c_{ij} f(i) f(j)$$

is maximized, we may try to find overlapping subsets $X_1, X_2 \subseteq X$ with $X_1 \cup X_2 = X$ and with an intersection $Z := X_1 \cap X_2$ such that $c_{ij} = c_{ji} = 0$ for all $i \in X_1 \backslash Z$ and $j \in X_2 \backslash Z$ in which case we have

$$\max \Big( C(f) \; : \; f \in \{\pm 1\}^X \Big) =$$

$$\max\left(C_1(f^*)+C_2(f^*)+\sum_{i,j\in Z} c_{ij}f^*(i)f^*(j)\ :\ f^*\in\{\pm 1\}^Z\right)$$

where for any $f^*\in\{\pm 1\}^Z$ the values $C_1(f^*)$ and $C_2(f^*)$ are defined in terms of the following optimization problems: put

$$c^1_{ij}:=\begin{cases} c_{ij} & \text{if } i,j\in X_1 \text{ and } \{i,j\}\not\subseteq Z\\ 0 & \text{if } i,j\in Z\end{cases}$$

and

$$c^2_{ij}:=\begin{cases} c_{ij} & \text{if } i,j\in X_2 \text{ and } \{i,j\}\not\subseteq Z\\ 0 & \text{if } i,j\in Z\end{cases}$$

and define $C_\alpha(f^*)$ $(\alpha=1,2)$ by

$$C_\alpha(f^*):=\max\left(\sum_{i,j\in X_\alpha} c^\alpha_{ij}f^*_\alpha(i)f^*_\alpha(j)\right)$$

where $f^*_\alpha$ runs over all extensions of $f^*$ to $X_\alpha$, that is, over all maps from $X_\alpha$ into $\{\pm 1\}$ with $f^*_\alpha|_Z=f^*$.

While this gives a handle to actually solve the original optimization problem exactly provided that can be done with the three resulting smaller problems, it for sure reduces the original search space of cardinality $2^{\#X}$ to a search space of cardinality $2^{\#Z}$ where for each point in that small search space two search spaces of cardinality $2^{\#(X_1\setminus Z)}$ and $2^{\#(X_2\setminus Z)}$, respectively, need to be investigated separately so that altogether the trivial upper bound $2^{\#X}$ for the time complexity of the original problem can be replaced by the number $2^{\#Z}(2^{\#(X_1\setminus Z)}+2^{\#(X_2\setminus Z)})=2^{\#X_1}+2^{\#X_2}$ of all points in the total "fibered" search space

$$\bigcup_{f^*\in\{\pm 1\}^Z}\left(\left\{f^*_1\in\{\pm 1\}^{X_1}\ :\ f^*_1|_Z=f^*\right\}\cup\left\{f^*_2\in\{\pm 1\}^{X_2}\ :\ f^*_2|_Z=f^*\right\}\right).$$

Yet, even if the three smaller problems cannot be solved exactly, searching heuristically for maps $f^*:Z\to\{\pm 1\}$ with a large value of $\sum_{i,j\in Z}c_{ij}f^*(i)f^*(j)$ and then extending them heuristically to maps $f^*_\alpha: X_\alpha\to Y$ with large values of

$$\sum_{i,j\in X_\alpha} c^\alpha_{ij}f^*_\alpha(i)f^*_\alpha(j)$$

might lead to good, if not optimal solutions of the original problem (and this applies even if the matrix entries $c_{ij}$ with $\{i,j\}\not\subseteq X_1$ and $\{i,j\}\not\subseteq X_2$ are very small compared to the other ones but not necessarily equal to 0).

Clearly, this idea is the starting point for many dynamic programming solutions of complex problems, e.g. the *spin-glass optimization* problem or the (closely related) so-called *small parsimonious tree* problem (cf. [12],[13],[14]).

Analysing the idea a bit more systematically suggests to look for a good concept of *embedding complexity*, that is a concept which measures how intricately a given set $S=S(\mathcal{X})$ is *embedded* into a large product set $\prod_{i\in\mathcal{X}}\mathcal{Y}_i$ – with a generally large index set $\mathcal{X}$

and generally small $\mathcal{Y}_i$ – by a family of maps $p_i : S \to \mathcal{Y}_i$ $(i \in \mathcal{X})$ or, more precisely, how easily the image $S(\mathcal{X})$ of the set $S$ can be described in terms of its projections $S(\mathcal{X}')$ onto smaller product sets $\prod\limits_{i \in \mathcal{X}'} \mathcal{Y}_i$ where $\mathcal{X}'$ runs through appropriate – and hopefully quite small – subsets of $\mathcal{X}$.

In the above example, a natural choice is

$$\mathcal{X} = \mathcal{X}(X, C) := \{(i,j) \in X^2 \ : \ c_{ij} \neq 0\}$$

for the index set $\mathcal{X}$ and

$$\mathcal{Y}_{(i,j)} := \{\pm 1\}^{\{i,j\}}$$

for the individual factor sets $\mathcal{Y}_{(i,j)}$ $(\,(i,j) \in \mathcal{X})$ into which the search space $S := \{\pm 1\}^X$ is projected via restriction by

$$p_{(i,j)} : S \to \mathcal{Y}_{(i,j)} : \quad f \mapsto f \ : \ _{\{i,j\}}.$$

Clearly, defining weight functions

$$w_{(i,j)} : \mathcal{Y}_{(i,j)} \to \mathbb{R}$$

by

$$w_{(i,j)}(g) := c_{ij} g(i) g(j) \qquad (g \in \mathcal{Y}_{(i,j)})$$

on every factor set $\mathcal{Y}_{(i,j)}$, the quadratic function $C(f)$ can now be expressed "linearly" as a simple sum

$$\sum_{(i,j) \in \mathcal{X}} w_{(i,j)} \left(p_{(i,j)}(f)\right)$$

of the weights of its projections and could be maximized by independent maximization in each component if the image $S(\mathcal{X})$ of $S$ would coincide with the full product set $\prod_{(i,j) \in \mathcal{X}} \mathcal{Y}_{(i,j)}$. Yet, even if this ideal situation is not provided by the given data, the above assumption $c_{ij} = 0$ unless $\{i,j\} \subseteq X_1$ or $\{i,j\} \subseteq X_2$ implies that an element $\left(g_{(i,j)}\right)_{(i,j) \in \mathcal{X}}$ from $\prod_{(i,j) \in \mathcal{X}} \mathcal{Y}_{(i,j)}$ is in $S(\mathcal{X})$ if and only if its two projections onto $\prod_{(i,j) \in \mathcal{X}_1} \mathcal{Y}_{(i,j)}$ and $\prod_{(i,j) \in \mathcal{X}_2} \mathcal{Y}_{(i,j)}$ – with $\mathcal{X}_\alpha := \mathcal{X}(X_\alpha, C|_{X_\alpha \times X_\alpha}) = \{(i,j) \in \mathcal{X} \ : \ \{i,j\} \subseteq X_\alpha\}$ $(\alpha = 1, 2)$ – both are contained in the correspondingly defined subsets $S(X_1)$ and $S(X_2)$. And it is exactly this simple fact regarding the embedding of $S = \{\pm 1\}^X$ into the product $\prod_{(i,j) \in \mathcal{X}} \mathcal{Y}_{(i,j)}$ on which the above proposal for reducing the given optimization problem to a family of considerably less complex problems is based.

Similarly and more generally, associating to any (simple) graph $\Gamma = (V, E)$ with vertex set $V$ and edge set $E \subseteq \{e \subseteq V \ : \ \#e = 2\}$, the embedding of $S := \{\pm 1\}^V$ into the product $\prod_{e \in E} \{\pm 1\}^e$ given, as above, by restriction $p_e : S \to \{\pm 1\}^e : \ f \mapsto f|_{\{e\}}$ $(e \in E)$, one sees easily that the *tree width* of $\Gamma$ can be invoked to provide a good measure for the complexity of that particular embedding and, hence, to provide means to solve problems related to graphs by dynamic programming procedures based on a Divide&Conquer strategy or to evaluate the efficiency of local optimization procedures.

Yet, we will not delve deeper into the abyss of abstract combinatorial complexity theory here. Rather, as promised in the title of this contribution, we will discuss two recent and rather successful applications of the Divide&Conquer principle in the molecular sciences – with the intention of (a) just demonstrating once more its wide range of applicability and introducing new fields of exploration and (b) of underlining the well-known facts that (bl) it is rarely clear at the beginning how to break up efficiently a given complex task into manageable subtasks, and that there is no routine all-purpose procedure of doing this systematically, and that (b2) even if an efficient way of doing this is anticipated, lots of additional efforts are needed to make such an idea really work.

The examples we want to discuss are the following two:

- a procedure for fast and complete enumeration of fullerene structures (cf. [23]) and,
- an algorithm for fast and reliable simultaneous alignment of sizeable families of biomolecular sequences.

The first example will demonstrate how the Divide&Conquer principle can be used to find efficiently *all* solutions of a complex problem by (a) first solving recursively a comparatively simple problem and (b) devising clever ways of glueing together appropriate pairs and triples of solutions of the simpler problem to find solutions of the original problem (and (c), of course, establishing beforehand theoretically that every solution can be constructed that way). The resulting computer program has already found many important applications in Carbon chemistry.

In the second example, the Divide&Conquer principle is used to generate heuristic (suboptimal) solutions for the task of aligning biomolecular RNA-, DNA- or amino-acid sequences so that phylogenetically and/or structurally corresponding sites in the individual sequences will be recognized by being assembled in just one column provided the given sequences are spelled out horizontally, one above the other. This is achieved by introducing *gaps* here or there into these sequences so as to make up for apparent inconsistencies between them, in particular to bring them all up to the same length, and to maximize overall similarity along the resulting columns. Sequence alignment is a fundamental task in string processing, and it is performed as a daily routine around the world in all computer laboratories servicing the molecular biosciences.

It should be noted that while these two algorithms still are quite conventional in that they are to be performed either by hand or, better, by a computer and not by the molecules themselves these algorithms are designed to analyse, it is not just a presently very fashionable and fund-raising idea to consider the molecular processes themselves from an algorithmic point of view, that is, as processes which actually perform more or less well-defined computations. Indeed, what we see happening presently in the molecular sciences is a continuously increasing intermingling of "wet" experimental and "dry" algorithmic approaches, each being used to drastically enhance and partly control the efficiency of the other, and we, the Mathematical Programming community, should better be aware (a) of the enormous potential *expansion* of the applicability of ideas relating traditionally to mathematical computer programming and (b) of the *changes* that that will require and bring about. This is evidenced for instance very clearly by the newly emerging field of *combinatorial* chemistry where Divide&Conquer strategies are implemented experimentally right at the heart of even the most basic experimental set-up.

# 2 Fullerene Structure Enumeration

From a purely combinatorial, graphtheoretical point of view, a fullerene isomer structure is a finite planar 3-regular graph of all whose faces are exclusively hexagons or pentagons (cf. Fig.1a). It follows easily from Euler's formula (in conjunction with standard bookkeeping devices) that any such graph must contain exactly 12 pentagons and that the number $n$ of its vertices and the number $N$ of its hexagons are related by the formula

$$n = 20 + 2 \cdot N.$$

Methods for reliable and efficient enumeration of fullerene isomers are presently a much discussed topic (cf. [22],[25],[28] [29],[37],[39],[3]). Most of the procedures applied so far use a *bottom-up* strategy: starting from a small subconfiguration, fullerene structures are generated by enlarging this subconfiguration stepwise in all conceivable ways (or in some particular ways assumed to be sufficient), e.g. by using one or the other variant of the so-called *spiral algorithm* (cf. [28],[39],[3]). These methods often meet prohibitive time constraints. So, quite a few implementations try to reduce complexity by shortcuts which then endanger reliability. Hence, none of these methods which is fast enough to be applicable for more than, say, 80 $C$-atoms can *guarantee* complete lists of fullerenes while those accepting possibly incomplete lists cannot go much beyond 100 C-atoms.

It may therefore be remarkable (cf. [6]) that a *top-down* Divide&Conquer strategy allows to design an algorithm for fullerene enumeration which is absolutely reliable – that is, it *guarantees* complete lists – and simultaneously amazingly efficient: On an HP9000/735, a complete enumeration of e.g. all $C_{60}$-structures (of which there are 1812) needs about 12 seconds: 6.5 seconds for the generation of sufficiently many such structures and 5.5 seconds for testing structural isomorphism. For fullerenes with about 100 atoms, the program appears to be faster by more than 6 orders of magnitude than previous (incomplete) ones.

In our algorithm, the Divide&Conquer strategy is applied using *Petrie paths* (cf. [9]) to reduce the problem of enumerating all fullerene structures with a given number of $C$-atoms to solving corresponding pairs or triplets of *PentHex Puzzles* (cf. Fig.1 and 2):

A *Petrie path* in a fullerene is a sequence of edges $e_1, e_2, \ldots, e_k$ such that any two consecutive edges $e_i, e_{i+1}$ $(i = 1, \ldots, k-1)$ share precisely one vertex (and, hence, they also share one face because the graph is 3-regular), while no face is shared by any three consecutive edges $e_i, e_{i+1}, e_{i+2}$ $(i = 1, \ldots, k-2)$. In other words, Petrie paths are *zig-zag* paths along the network of edges provided by a fullerene which, at each vertex they meet, turn right or left alternatively.

It is clear that for each pair $e_1, e_2$ of edges which share precisely one vertex and for each $k \geq 2$, there exists precisely one Petrie path $e_1, e_2, \ldots, e_k$ and that starting with an arbitrary such pair $e_1, e_2$, there must exist a smallest $k \geq 2$ such that the end vertex of $e_k$ coincides with one of the vertices which have been met before. In Fig. 1, this vertex is indicated by a full circle. If this is the vertex of $e$ where our Petrie path started and if $e_{k-1}, e_k$, and $e_1$ do not share a face, we have a *closed Jordan* Petrie path which cuts our spherical fullerene structure into two hemispheres, both of which have a zig-zag boundary consisting of precisely $k$ edges (see Fig.1a). Otherwise, we may reverse our direction and follow the reverse Petrie path starting with $e_2$ and then continuing with $e_1, e_0, e_{-1}, \ldots, e_{-l}$ until again, for some $l \geq 0$, we meet some vertex visited before (including, of course, the vertices of $e_1, e_2, \ldots, e_k$), indicated by an open circle. In this case the total path $e_{-l}, e_{-l+1}, \ldots, e_0, e_1, \ldots, e_k$ cuts our spherical fullerene structure in precisely three *patches*, either of the form depicted in Fig.1b or of that depicted in Fig.1c.

In each case, the boundaries of these patches are again zig-zag paths (that is, the *third* edges emerging from the vertices along the boundary – those which are *not* followed by our path – alternatively stick out of and into the patch) except for at most two localities – involving the vertices where our Petrie path met itself in cases 2 and 3 – where at least two consecutive vertices at the boundary have their third edge sticking out. Clearly, once we know those (either two or three) whole patches, we can glue them together appropriately to regain our fullerene.

It remains to discuss how the structure of these patches can be (re-)constructed. This leads to the concept of *PentHex Puzzles*: A PentHex Puzzle is given by separating a given finite set $S$ of points – called boundary vertices – on a circle into two disjoint subsets, say $A$ and $B$. The associated task is to (re)create *fullerene patches* by filling the disc inside the circle by a planar graph so that this graph contains the circle line, its vertices on the circle line are precisely the points in $S$ (that is, the boundary vertices), all of its vertices except those in $A$ have degree 3 while those in $A$ have degree 2, and all of its faces are exclusively pentagons or hexagons. Invoking Euler's formula again, one easily sees that the number of pentagonal faces in any such patch equals $6 + \#B - \#A$ – so, we must have $\#A - \#B \leq 6$. Obviously, any of the above patches is a solution of the PentHex Puzzle defined by its boundary line, with $A$ the set of vertices along the boundary where the third edge sticks out of and $B$ those where it sticks into the patch. Moreover, the way our patches were constructed in terms of Petrie paths ensures that only *convex* PentHex Puzzles have to be solved, that is, those where no two consecutive vertices are in $B$.

In cases like that depicted in Fig.1a, that is if $\#A = \#B$, the structure of such a puzzle can be encoded by just one number $M$ $(= \#A = \#B)$.

If $\#A > \#B$, the edges with both incident vertices in $A$ divide the boundary into $j := \#A - \#B > 0$ segments. In this case, the sequence $(M_1, M_2, \ldots, M_j)$, denoting consecutively the numbers of vertices in $B$ in these segments, can be used to encode the structure of the puzzle (cf. Fig.2).

Fortunately, *convex* PentHex Puzzles can be solved recursively quite easily. In case $\#A = \#B =: M$, any solution starts with a given number $H \geq 0$ of inscribed hexagon circles, each of length $M$ and each reproducing the given puzzle in its interior (cf. Fig.3a), until the first pentagons are met in which case removing the next inscribed circle of pentagons and hexagons leads to a PentHex Puzzle of type $(M_1, \ldots, M_j)$ with $j$ the number of pentagons in that circle (and $M = j + M_1 + \ldots + M_j$). And in case $\#A > \#B$, we can start at an edge with both of its vertices in $A$ and proceed either to the next such edge (as indicated in Fig.3b) or until we meet a pentagon (Fig. 3c). Convexity guarantees that this will never interfere with other parts of the boundary of the given patch. This way, the PentHex Puzzle can be reduced to one with a smaller number of $B$-type vertices. The inverse of the described reduction process can be used to construct all solutions of convex PentHex Puzzles with – in principle – any given number of hexagons, starting from a pentagon or a hexagon.

| A | B | A | C | A | B | A | C |
|---|---|---|---|---|---|---|---|
| | | 60: | 1 | 88: | 81 738 | 136: | 79 362 |
| | | 62: | 0 | 90: | 99 918 | 138: | 98 541 |
| | | 64: | 0 | 92: | 126 409 | 140: | 121 354 |
| | | 66: | 0 | 94: | 153 493 | 142: | 151 201 |
| 20: | 1 | 68: | 0 | 96: | 191 839 | 144: | 186 611 |
| 22: | 0 | 70: | 1 | 98: | 231 017 | 146: | 225 245 |
| 24: | 1 | 72: | 1 | 100: | 285 914 | 148: | 277 930 |
| 26: | 1 | 74: | 1 | 102: | 341 658 | 150: | 335 569 |
| 28: | 2 | 76: | 2 | 104: | 419 013 | 152: | 404 667 |
| 30: | 3 | 78: | 5 | 106: | 497 529 | 154: | 489 646 |
| 32: | 6 | 80: | 7 | 108: | 604 217 | 156: | 586 264 |
| 34: | 6 | 82: | 9 | 110: | 713 319 | 158: | 697 720 |
| 36: | 15 | 84: | 24 | 112: | 860 161 | 160: | 836 497 |
| 38: | 17 | 86: | 19 | 114: | 1 008 444 | 162: | 989 495 |
| 40: | 40 | 88: | 35 | 116: | 1 207 119 | 164: | 1 170 157 |
| 42: | 45 | 90: | 46 | 118: | 1 408 553 | 166: | 1 382 953 |
| 44: | 89 | 92: | 86 | 120: | 1 674 171 | 168: | 1 628 029 |
| 46: | 116 | 94: | 134 | 122: | 1 942 929 | 170: | 1 902 265 |
| 48: | 199 | 96: | 187 | 124: | 2 295 721 | 172: | 2 234 133 |
| 50: | 271 | 98: | 259 | 126: | 2 650 866 | 174: | 2 601 868 |
| 52: | 437 | 100: | 450 | 128: | 3 114 236 | 176: | 3 024 383 |
| 54: | 580 | 102: | 616 | 130: | 3 580 637 | 178: | 3 516 365 |
| 56: | 924 | 104: | 823 | 132: | 4 182 071 | 180: | 4 071 832 |
| 58: | 1 205 | 106: | 1 233 | 134: | 4 787 715 | 182: | 4 690 880 |
| 60: | 1 812 | 108: | 1 799 | 136: | 5 566 948 | 184: | 5 424 777 |
| 62: | 2 385 | 110: | 2 355 | 138: | 6 344 698 | 186: | 6 229 550 |
| 64: | 3 465 | 112: | 3 342 | 140: | 7 341 204 | 188: | 7 144 091 |
| 66: | 4 478 | 114: | 4 468 | 142: | 8 339 033 | 190: | 8 187 581 |
| 68: | 6 332 | 116: | 6 063 | 144: | 9 604 410 | 192: | 9 364 975 |
| 70: | 8 149 | 118: | 8 148 | 146: | 10 867 629 | 194: | 10 659 863 |
| 72: | 11 190 | 120: | 10 774 | 148: | 12 469 092 | 196: | 12 163 298 |
| 74: | 14 246 | 122: | 13 977 | 150: | 14 059 173 | 198: | 13 809 901 |
| 76: | 19 151 | 124: | 18 769 | 152: | 16 066 024 | 200: | 15 655 672 |
| 78: | 24 109 | 126: | 23 589 | 154: | 18 060 973 | 202: | 17 749 388 |
| 80: | 31 924 | 128: | 30 683 | 156: | 20 558 765 | 204: | 20 070 486 |
| 82: | 39 718 | 130: | 39 393 | 158: | 23 037 593 | 206: | 22 606 939 |
| 84: | 51 592 | 132: | 49 878 | 160: | 26 142 839 | 208: | 25 536 557 |
| 86: | 63 761 | 134: | 62 372 | 162: | 29 202 540 | 210: | 28 700 677 |

Table 1: Numbers of fullerenes and IPR-fullerenes, with numbers of fullerenes with $n$ and IPR-fullerenes with $n+48$ atoms in one row. A stands for the number of vertices, B for the number of fullerenes and C for the number of IPR-fullerenes.

By using an appropriate form of dynamic programming – and lots of care in setting it up explicitly, that is, in implementing it (using a sophisticated lexicographic coding method to also make sure that the resulting list of fullerene structures never contains two structurally isomorphic copies, thereby taking orientation either into account or neglecting it), *and* in using the memory –, we have computed all fullerene structures for up to $n = 170$ atoms, as well as for up to $n = 214$ atoms those special structures which obey the *isolated pentagon rule* (IPR) – that is, structures where every pentagon is surrounded by hexagons, only.

Table 1 records the number of structures we have found. It seems remarkable that for $n \geq 38$ the number $F(n)$ of all fullerene isomers with $n$ atoms roughly coincides with the number $F_{IPR}(n+48)$ of all IPR-fullerene isomers with $n+48$ atoms, and that for $n$ divisible by 4 the difference between $F(n)$ and $F(n-2)$ roughly coincides with the difference between $F(n+2)$ and $F(n)$.

The method extends easily to fullerene-like molecular cage (and the related patch) structures which include quadrangles and/or triangles and it can also be applied (though not simply) to handle those which include heptagons etc. [19] as well as to 4-regular planar graphs or even some types of regular graphs of higher genus and further related problems (cf. [4],[5]).

To summarize: Given any fullerene structure, that structure can be broken up into two or three *convex fullerene patches* in several, yet not too many distinct ways, lists of possible patches can be generated recursively quite easily by solving the associated PentHex Puzzles, and the given – as well as any other such – structure can thus be found by glueing together appropriate pairs and triples of patches in all legitimate ways.

This is surely a not quite canonical way of applying the Divide&Conquer principle which is generally used to find one structure, map (or whatever) out of a virtually very large list of such structures, maps (or whatever) rather than to create such a list; yet, it has proven to be an amazingly efficient application of that principle, speeding up the enumeration process by quite a few orders of magnitude compared with competing solutions – and adding *reliability* regarding the completeness of the lists of obtained structures as an extra bonus.

# 3 A Fast and Reliable Method for Simultaneous Multiple Sequence Alignment

## 3.1 The Alignment Problem

Our second example is a new algorithm for producing close to optimal solutions of the multiple sequence alignment problem, called the *Divide&Conquer Alignment* algorithm (DCA, [40]). Multiple sequence alignment is a well-studied but still not satisfactorily solved problem in string processing having its most important application in computational molecular biology. Indeed, many important conclusions to be drawn from the sequence of *residues* in a big biomolecule, that is, of amino acids in a given protein or that of nucleotides in a given RNA or DNA molecule, depend crucially on *comparing* that sequence with other such sequences by means of appropriately constructed *alignments.* For example, such alignments are used to detect homologues among sequences in genome databases, to study phylogenetic relationships, or to identify structurally or functionally important parts of the molecule in question.

Consequently, establishing *fast and reliable tools for sequence alignment* is one of the most fundamental tasks in present day computational biology, enjoying an abundance of publications and software contributions (see [32], [8], or [48]).

The overall strategy one has to follow for producing reliable alignments is quite obvious: by inserting *gaps* here and there into the sequences one wants to align, one tries to come up with sequences of equal length so that the sequence entries at each site – that is, in each column when the (aligned) sequences themselves are spelled out horizontally, one below the other – exhibit a biologically meaningful diversity, possibly of not too large a degree, which can be interpreted in a coherent way. For example, one may head for a phylogenetic interpretation implying that the sequence entries at a given site have evolved from a common ancestor entry, or for a structural interpretation implying that the aligned residues are placed at similar locations within the folded molecule.

Consequently, because it is the *similarity* of sequence patterns which is supposed to signal phylogenetic and/or structural kinship between the sequences, the aim of sequence-alignment procedures is to *maximize* overall similarity. Thus, all that is required is

- specifying in a quantifyable way the term *overall similarity*, and
- constructing algorithms which produce alignments which maximize that overall similarity or, if this turns out to be too time consuming, at least exhibit a rather high degree of that similarity.

While the first task needs input from biology as well as from mathematical modelling, the second task is a purely mathematical one. Unfortunately, many ideas relating to the first task cannot be tested, and important structural parameters suggested by these ideas cannot be evaluated easily unless the second task has been dealt with appropriately.

To tackle that second task, the starting point is clearly to find good methods for aligning *two* sequences – that is, for *pairwise* alignment – and algorithms for solving this problem were developed successfully already a quarter of a century ago ([33],[47]). These algorithms follow the well known *dynamic programming* method. However, their natural and straightforward generalizations to three or more sequences (together with the natural extension of quantifying overall similarity in terms of the so-called *sum-of-pairs score*, see below) quickly run into prohibitive memory and time constraints as number or length of sequences increase. Therefore, almost all techniques for aligning larger sets of sequences are based on first performing a series of pairwise alignments (using, if necessary, appropriate adaptations of the standard algorithm aligning *profiles* of sequences rather than sequences) and then constructing a multiple alignment in a "hill-climbing" manner (see for example [11],[30] for reviews). However, these methods (e.g. CLUSTAL [44], DFALIGN [16], GENALIGN [27], MULTAL [43]) though fast, can be used with some reservation only for the following two reasons: they easily run into local, but not necessarily global optima, – a risk, which is inherent in any hill-climbing method – and they often do not even accept a well-defined optimality criterion for multiple sequence alignment.

In order to circumvent these problems, one has to stick to the original task of trying to construct high quality *simultaneous alignments.* In the late eighties, significant progress with this technique was made by CARRILLO and LIPMAN [7]: it became possible to align simultaneously and optimally up to between six and eight protein sequences (of medium length and comparatively high pairwise similarity) in some minutes. This was achieved by cutting down the (high-dimensional) search space used in dynamic programming, by considering projections of precalculated heuristic alignments onto the (two-dimensional) "boundaries" of that space. Yet, even when implementing this idea using highly sophisticated implementation techniques, the resulting program, called MSA, often requires more time and/or memory space than available when ever it has to deal with larger data sets [18].

Hence, for dealing with such cases, we propose a new procedure based on a simple but amazingly efficient heuristics which we have called the *Divide&Conquer Alignment* algorithm, DCA ([15],[45],[42]).

The general idea of DCA is rather simple: Each of the sequences is cut in two by cutting it just behind a suitable *slicing site* somewhere close to its midpoint. This way, the problem of aligning one family of (long) sequences is divided into the two problems of aligning two families of (shorter) sequences, the prefix and the suffix sequences. This procedure is re-iterated until the sequences are sufficiently short so that they can be aligned optimally by MSA. Finally, the resulting short alignments are concatenated, yielding a multiple alignment of the original sequences.

Of course, the main difficulty with this approach is how to identify those slicing-site combinations which lead to an optimal or – at least – close to optimal alignment. Here, a heuristic based on so-called *secondary-charge matrices* which are used for quantifying the compatibility of slicing sites in distinct sequences proved to be successful. Several ways of speeding up the search for these slicing sites are possible some of which are implemented already. They also will be discussed below.

## 3.2 A Formal Set-Up: The Weighted Sum of Pairs Score for Multiple Sequence Alignment

Next, we define multiple alignments formally and we describe the basic principles of evaluating quantitatively the quality of a given multiple alignment (for further reference see [11],[38], and [48]).

Suppose that we are given a family $S = (\mathbf{s}_1, \ldots, \mathbf{s}_k)$ of $k$ sequences:

$$\begin{array}{rcl} \mathbf{s}_1 & = & s_{11}\, s_{12} \ldots s_{1n_1} \\ & \vdots & \\ \mathbf{s}_k & = & s_{k1}\, s_{k2} \ldots s_{kn_k} \end{array}$$

where each sequence entry $s_{ij}$ represents a letter from a given finite alphabet $\mathcal{A}$. An *alignment* of the sequences $S$ is a matrix $M = (m_{ij})_{1 \le i \le k, 1 \le j \le N}$ where

- $m_{ij} \in \mathcal{A} \cup \{-\}$, with '$-$' denoting the *gap letter* supposed not to be contained in $\mathcal{A}$,

- the rows $\mathbf{m}_l := m_{l1} \ldots m_{lN}$ of $M$ considered as sequences of symbols from $\mathcal{A} \cup \{-\}$, reproduce the sequences $\mathbf{s}_l$ upon elimination of the gap letters ($1 \le l \le k$),

- the matrix $M$ has no column, only containing gaps.

For example, one alignment of $S = \{\mathbf{s}_1, \mathbf{s}_2\}$ with $\mathbf{s}_1 =$ GTATGCCG and $\mathbf{s}_2 =$ GTGTCGG is given by the matrix

$$M := \begin{pmatrix} \mathtt{G\,T\,A\,T\,G\;C\,C\,G\,-} \\ \mathtt{G\,T\,G\,T\,-\,-\,C\,G\;G} \end{pmatrix},$$

and another one is given by

$$M' := \begin{pmatrix} \mathtt{G\;T\;A\;T\,G\,C\,C\,-\,G} \\ \mathtt{G\,-\,-\,T\,G\,T\,C\;G\;G} \end{pmatrix}.$$

We denote the set of all alignments of $S$ by $M_S$. Assume that we are given a pairwise distance

$$d : (\mathcal{A} \cup \{-\})^2 \to \mathbb{R}$$

(the distances given e.g. by a substitution cost matrix (cf. [10]) with appropriately chosen gap penalties[1] $d(a,-)$ for all $a \in \mathcal{A}$). For each pair of rows $\mathbf{m}_p, \mathbf{m}_q$ in an alignment $M \in M_S$, define

$$w_{\mathbf{m}_p,\mathbf{m}_q} := \sum_{i=1}^{N} d(m_{pi}, m_{qi}),$$

and denote by $w_{opt}(\mathrm{s}_p, \mathrm{s}_q)$ the minimum of $w_{\mathbf{m}_p,\mathbf{m}_q}$, taken over all alignments $M$.

The *weighted sum of pairs score* for an alignment $M \in M_S$ relative to a given family of (generally non-negative) weight parameters $\alpha_{p,q}$ $(1 \leq p < q \leq k)$ is defined by

$$w(M) := \sum_{1 \leq p < q \leq k} \alpha_{p,q} \cdot w_{\mathbf{m}_p,\mathbf{m}_q}.$$

The *multiple alignment problem* that we aim to solve is to search for matrices $M \in M_S$ whose weighted sum of pairs score $w(M)$ is small.

The logic for introducing the weight parameters $\alpha_{pq}$ (from which procedures for choosing them appropriately are to be deduced) is the following one: In general, any set of related biological sequences contains some sequences which are more closely related to one another than to the remaining ones, and highlighting their similarity might often be more important than forcing them to independently conform to the patterns of the other sequences. On the other hand, as almost any sample of sequences is biased in one way or the other (even, most probably, the sample provided by Nature itself), a perhaps overrepresented subset of highly similar sequences in a data set should not be allowed through its sheer size to force all the other sequences to conform to its patterns. Boths goals, highlighting similarity between closely related sequences and discounting overrepresentation of certain subclasses of sequences can (hopefully) be achieved by choosing appropriate weight factors,– one might even consider using homology-dependent distance scores for each given pair of sequences.

As mentioned above, optimizing $w(M)$ can be solved in principle by straightforward *dynamic programming* (cf. [33, 38]). However, this is possible only in theory at present: in practice, the space and time requirements for dynamic programming, even in its most sophisticated forms, make it virtually impossible to deal with, say, five not *highly* homologous sequences of length approximately 1000. However, such tasks present themselves easily when dealing with problems from biological sequence analysis.

## 3.3 The Divide&Conquer Approach

How does the DCA procedure attack this problem? As mentioned already above, given a family of sequences $\mathrm{s}_1, \ldots, \mathrm{s}_k$ of length $n_1, \ldots, n_k$, respectively, each of these sequences is to be cut just behind an appropriately chosen slicing site somewhere near to its midpoint. This way, the original alignment problem is reduced to the two subproblems of aligning the two resulting families of prefix and suffix subsequences, respectively. These will be handled by the same procedure in a recursive manner. The recursion stops when

[1] More sophisticated gap-penalty functions have been considered, e.g. the so-called affine gap penalty, which works for pairwise as well as for multiple sequence alignment [1] [17].

a certain stopping criterion is fulfilled. The remaining subsequence families are then aligned by MSA [24],[18], and the resulting alignments are concatenated.

The main problem is to find a $k$-tuple of *ideal* slicing sites $(c_1, c_2, \ldots, c_k)$ (with $0 \leq c_p \leq n_p$ for $p = 1, \ldots, k$) so that the simple concatenation of the two optimal alignments of the prefix sequences $\mathrm{s}_1(\leq c_1), \mathrm{s}_2(\leq c_2), \ldots, \mathrm{s}_k(\leq c_k)$ and the suffix sequences $\mathrm{s}_1(> c_1), \mathrm{s}_2(> c_2), \ldots,$ $\mathrm{s}_k(> c_k)$ forms an optimal alignment of the original sequences[2].

Obviously, for any fixed site $\hat{c}_1$ $(0 \leq \hat{c}_1 \leq n_1)$, there exists a $(k-1)$-tuple of slicing sites $(c_2(\hat{c}_1), \ldots, c_k(\hat{c}_1))$, such that $(\hat{c}_1, c_2(\hat{c}_1), \ldots, c_k(\hat{c}_1))$ forms an optimal $k$-tuple. Unfortunately, finding exactly these sites requires approximately as much time as solving the original optimization problem directly. So, of course, this is not the method of choice.

Instead, we aim to find *C-optimal* slicing sites which can be computed in terms of pairwise sequence comparisons, only. More precisely, we use the dynamic programming procedure to compute, for all pairs of sequences $(\mathrm{s}_p, \mathrm{s}_q)$, and for all slicing sites $c_p$ of $\mathrm{s}_p$ and $c_q$ of $\mathrm{s}_q$ the *secondary charge* $C_{\mathrm{s}_p,\mathrm{s}_q}[c_p, c_q]$ defined by

$$C_{\mathrm{s}_p,\mathrm{s}_q}[\,c_p, c_q\,] \;:=\; w_{opt}\,(\,\mathrm{s}_p(\leq c_p), \mathrm{s}_q(\leq c_q)\,) + w_{opt}\,(\,\mathrm{s}_p(> c_p\,), \mathrm{s}_q(> c_q)\,) - w_{opt}\,(\,\mathrm{s}_p, \mathrm{s}_q\,)$$

which quantifies the additional charge imposed by forcing the alignment of $\mathrm{s}_p$ and $\mathrm{s}_q$ to optimally align $\mathrm{s}_p(\leq c_p)$ and $\mathrm{s}_q(\leq c_q)$ as well as $\mathrm{s}_p(> c_p)$ and $\mathrm{s}_q(> c_q)$, rather than aligning $\mathrm{s}_p$ and $\mathrm{s}_q$ optimally. The calculation of the matrices $C_{\mathrm{s}_p,\mathrm{s}_q}$ can be performed by computing *forward* and *reverse* matrices in a similar way as described in [21],[31], [46]. Note that there exists, for every fixed $\hat{c}_p$, at least one vertex $c_q(\hat{c}_p)$ with $C_{\mathrm{s}_p,\mathrm{s}_q}[\hat{c}_p, c_q(\hat{c}_p)] = 0$ which can be computed easily from any optimal *pairwise* alignment of $\mathrm{s}_p$ and $\mathrm{s}_q$. The problem *multiple* alignments have to face, is that $c_q\,(c_p(\hat{c}_1))$ might not coincide with $c_q\,(\hat{c}_1)$, that is, that given pairwise optimal alignments may be incompatible with each other - much in analogy to *frustrated systems* considered in statistical physics.

To search for good $k$-tuples of slicing sites, we define the *multiple additional charge* $C(c_1, \ldots, c_k)$ imposed by slicing the sequences at any given $k$-tuple of slicing sites $(c_1, \ldots, c_k)$ as a weighted sum of secondary charges over all projections $(c_p, c_q)$, that is, we put

$$C(c_1, c_2, \ldots, c_k) \;:=\; \sum_{1 \leq p < q \leq k} \alpha_{p,q} \cdot C_{\mathrm{s}_p,\mathrm{s}_q}[c_p, c_q],$$

where the $\alpha_{p,q}$ are the same sequence-dependent weight factors as above.

Our proposition is now that using as the preferred slicing-site combinations those *C-optimal* $k$-tuples that minimize – for a given fixed slicing site $\hat{c}_1$ of $\mathrm{s}_1$ – the value $C(\hat{c}_1, c_2, \ldots, c_k)$ over all slicing sites $c_2, \ldots, c_k$ of $\mathrm{s}_2, \ldots, \mathrm{s}_k$, respectively, will result in very good, if not optimal multiple alignments because, this way, the mutual *frustration* is distributed as fairly as possible.

In conclusion, this leads to the following general procedure:

```
Algorithm $DCA\,(\mathrm{s}_1, \mathrm{s}_2, \ldots, \mathrm{s}_k, L)$

If $\min_{i \in \{1,2,\ldots,k\}}\{n_i\} \leq L$
```

[2] Here, $\mathrm{s}_p(\leq c_p)$ denotes the prefix subsequence of $\mathrm{s}_p$ with indices running from 1 to $c_p$ and $\mathrm{s}_p(> c_p)$ denotes the suffix subsequence of $\mathrm{s}_p$ with indices running from $c_p + 1$ to $n_p$, $1 \leq p \leq k$.

```
then  return the optimal alignment of $s_1, s_2, \ldots, s_k$ (using e.g. MSA);
else  return the concatenation of
                  $DCA(s_1(\leq c_1), s_2(\leq c_2), \ldots, s_k(\leq c_k), L)$
              and $DCA(s_1(> c_1), s_2(> c_2), \ldots, s_k(> c_k), L)$;
      where  $(c_1, c_2, \ldots, c_k) := calc\text{-}cut(s_1, s_2, \ldots, s_k)$.
```

In the following section, we describe how to realize the subroutine *calc-cut* which computes a $k$-tuple of $C$-optimal slicing sites.

## 3.4 Efficiently Calculating the Slicing Sites

In a naive implementation, the search *calc-cut* for $C$-optimal slicing sites $(c_1, c_2, \ldots, c_k)$ needs time $\mathcal{O}(k^2n^2 + n^{k-1})$, where $n$ is the length of the longest of the sequences $s_1, s_2, \ldots, s_k$: the computation of all pairwise secondary matrices takes $\mathcal{O}(k^2n^2)$ time and, for given $\hat{c}_1$, all possible combinations of $c_2, \ldots, c_k$ have to be checked to find the tuple that minimizes $C$ in altogether $\mathcal{O}(n^{k-1})$ single steps.

We reduce this running time and the required memory (which is of order $\mathcal{O}(k^2n^2)$ for the naive version) by first precalculating an estimate $\widehat{C}$ for

$$C_{opt}(\hat{c}_1; s_1, \ldots, s_k) := \min_{c_2,\ldots,c_k} C(\hat{c}_1, c_2, \ldots, c_k).$$

This allows us to prune the search space enormously: Because the multiple additional cost $C(\hat{c}_1, c_2, \ldots, c_k)$ is defined as a sum of non-negative numbers, it is possible to exclude any tuple of slicing sites $(\hat{c}_1, c_2, \ldots, c_k)$, whenever one of the summands is larger than the minimum $\widehat{C}$ found so far. In particular, for fixed $\hat{c}_1$, no $c_q$ with $\alpha_{1,q} \cdot C_{s_1,s_q}[\hat{c}_1, c_q] \geq \widehat{C}$ can ever lead to a smaller sum $C$.

With this in mind, a $C$-optimal tuple of slicing sites can be calculated as follows:

**Function** *calc-cut* ( $s_1, s_2, \ldots, s_k$ )

1. Reorder $s_1, s_2, \ldots, s_k$ so that $s_1$ is the longest of all sequences in question.
2. Fix $\hat{c}_1 := \lceil \frac{n_1}{2} \rceil$;
3. calculate and save rows $row^{\hat{c}_1}_{1,q}[j] := C_{s_1,s_q}[\hat{c}_1, j] \quad (2 \leq q \leq k, 1 \leq j \leq n_q)$;
4. locate slicing sites $\hat{c}_2, \ldots, \hat{c}_k$ such that $row^{\hat{c}_1}_{1,q}[\hat{c}_q] = 0 \quad (2 \leq q \leq k)$;
5. calculate the estimate [3]
$$\widehat{C} := \sum_{1 \leq p < q \leq k} \alpha_{p,q} \cdot C_{s_p,s_q}[\hat{c}_p, \hat{c}_q] = \sum_{2 \leq p < q \leq k} \alpha_{p,q} \cdot C_{s_p,s_q}[\hat{c}_p, \hat{c}_q],$$
6. Calculate lower and upper bounds $l_q$ and $u_q$ such that $\alpha_{1,q} \cdot row^{\hat{c}_1}_{1,q}[j] \geq \widehat{C}$ for all $j < l_q$ and for all $j > u_q$ $(2 \leq q \leq k)$. The intermediate segment $row^{\hat{c}_1}_{1,q}[l_q], \ldots, row^{\hat{c}_1}_{1,q}[u_q]$ forms the *relevant part* of each row $row^{\hat{c}_1}_{1,q}$.

[3] Some additional approaches have been developed which work by using $(\hat{c}_1, \hat{c}_2, \ldots, \hat{c}_k)$ as the starting points and then sucessively minimize $C(\hat{c}_1, c_2, \ldots, c_k)$ over some $c_i$ $(i \in \{2, \ldots, n\})$ while keeping the other slicing sites $c_j$ $(j \neq i)$ fixed. This leads to significantly smaller estimates for $C$ than the one described above [35].

7. Given these bounds, compute and save the *relevant parts* of the matrices $C_{s_p,s_q}$, defined by $C_{s_p,s_q}[c_p, c_q]$, with $l_p \leq c_p \leq u_p$ and $l_q \leq c_q \leq u_q$.

8. Search for better slicing-site combinations $(\hat{c}_1, c_2, \ldots, c_k)$ within the relevant parts of the rows $row_{1,q}^{\hat{c}_1}$ and the matrices $C_{s_p,s_q}$. Thereby, the sum $C$ can be computed step by step and the search can be stopped, if an intermediate result is larger than $\widehat{C}$.

During this final search, better values for $\widehat{C}$ may be obtained, too, so that, with decreasing values of $\widehat{C}$, the relevant part of the rows $row_{1,q}^{\hat{c}_1}$ can possibly be further reduced, diminishing the search space even more.

Obviously, the worst case time and space complexity of this approach still remains of the order of $\mathcal{O}(k^2n^2+n^{k-1})$ and $\mathcal{O}(k^2n^2)$, respectively, for the (very improbable) case that the bounds $l_i$ and $u_i$ can never be increased or decreased, respectively. But for biological sequences, the effect is enormous: For calculating the first tuple of slicing sites in the recursion (which takes far the longest time of all slicing-site computations) replacing $n$ by the length $r := \max_{p=2,\ldots,k}\{u_p - l_p + 1\}$ of the longest of the remaining relevant parts of the rows, usually results in a reduction of at least two orders of magnitude per sequence (for small $k$), yielding memory savings for the matrices by several orders of magnitude and reducing the expected time and space complexity to $\mathcal{O}(k^2n^2 + r^{k-1})$ and $\mathcal{O}(kn + k^2r^2)$, respectively, (cf. [41]).

## 3.5 Further Improvements

To speed up the procedure for still larger $k$, an additional preprocessing step can be used: To determine, say, the slicing site $c_k$ of $s_k$, the optimal additional charge of any subfamily $s'_1 := s_1, s'_2, \ldots, s'_\kappa$ of our sequences not including sequence $s_k$ can be used to compute a better estimate for those values of $C_{s_1,s_k}[\hat{c}_1, c_k]$ which need further consideration: Clearly, for every slicing site $c_k$ of $s_k$ we have

$$C_{opt}(\hat{c}_1; s'_1, \ldots, s'_\kappa) + \alpha_{1,k} \cdot C_{s_1,s_k}[\hat{c}_1, c_k] \leq C(\hat{c}_1, c_2, \ldots, c_k)$$

for all possible slicing sites $c_2, \ldots, c_{k-1}$ of $s_2, \ldots, s_{k-1}$. Hence, we can exclude a slicing site $c_k$ from further consideration if for some such family $s'_1, s'_2, \ldots, s'_\kappa$ and some upper bound $\widehat{C}$ of $C_{opt}(\hat{c}_1; s_1, \ldots, s_k)$, we have

$$\alpha_{1,k} \cdot C_{s_1,s_k}[\hat{c}_1, c_k] \geq \widehat{C} - C_{opt}(\hat{c}_1; s'_1, \ldots, s'_\kappa),$$

as this implies $\widehat{C} \leq \widehat{C}(\hat{c}_1, c_2, \ldots, c_k)$ for all slicing sites $c_2, \ldots, c_{k-1}$ of $s_2, \ldots, s_{k-1}$.

Using this principle to its fullest extent would require computing $3 = \binom{3}{2}$ times the optimal slicing sites for three sequences in case of altogether four sequences, $6 = \binom{4}{2}$ times the optimal slicing sites for three and – based on that – $4 = \binom{4}{3}$ times the optimal slicing sites for four sequences in case of altogether five sequences, and so on. Hence, for $k$ sequences,

$$2^{k-1} - k - 1 = \binom{k-1}{2} + \binom{k-1}{3} + \ldots + \binom{k-1}{k-2}$$

additional optimal slicing-site combinations for $3, 4, \ldots, k-1$ sequences would have to be computed. Clearly, this would always be worth the trouble if the average rate $\alpha$ by which – at each step – the average length $a$ of the relevant part of each sequence in question is reduced, is just $(a-1)/a$: indeed, if $\alpha < (a-1)/a$, then

$$\sum_{i=2}^{k-2}\binom{k-1}{i}(\alpha^{i-1}\cdot a)^i < \sum_{i=2}^{k-2}\binom{k-1}{i}(\alpha\cdot a)^i < (1+\alpha\cdot a)^{k-1} \leq a^{k-1}.$$

In addition, computing the optimal slicing-site combination $\hat{c}_1, \ldots, \hat{c}_\kappa$ for a fixed slicing site $\hat{c}_1$ of $\mathrm{s}_1$ and any given sequence family $\mathrm{s}'_1 := \mathrm{s}_1, \mathrm{s}'_2, \ldots, \mathrm{s}'_\kappa$ may also help to improve the estimates $\widehat{C}$ for $C_{opt}(\hat{c}_1; \mathrm{s}'_1, \ldots, \mathrm{s}'_\kappa, \mathrm{s}'_{\kappa+1})$ for each new sequence $\mathrm{s}'_{\kappa+1}$ in view of

$$C_{opt}(\hat{c}_1; \mathrm{s}'_1, \ldots, \mathrm{s}'_\kappa, \mathrm{s}'_{\kappa+1}) \leq C_{opt}(\hat{c}_1; \mathrm{s}'_1, \ldots, \mathrm{s}'_\kappa) + \sum_{p=1}^{\kappa} \alpha_{p,\kappa+1} C_{\mathrm{s}'_p,\mathrm{s}'_{\kappa+1}}[\hat{c}_p, c_{\kappa+1}]$$

for each slicing site $c_{\kappa+1}$ of $\mathrm{s}'_{\kappa+1}$.

And finally, the variety of optimal slicing sites coming up in such a computation may also be useful for evaluating the rate of mutual frustration as well as the reliability and the quality of the slicing sites finally chosen and of the alignment(s) resulting from that choice.

A different approach to the time problem (which can, of course, be combined with the one outlined just above) is motivated by the regular shape of the matrices $C_{\mathrm{s}_p,\mathrm{s}_q}$. It utilizes the observation that the entries in a row of a secondary charge matrix from left to right generally start with rather high values, decrease almost monotonically for a long time, reach the value zero at the slicing site corresponding to an optimal pairwise alignment, and then increase almost monotonically again to high values. Thus, for each row $i$, $l_p \leq i \leq u_p$, of the secondary charge $C_{\mathrm{s}_p,\mathrm{s}_q}$, we can define lower and upper *monotony bounds* $L_{\mathrm{s}_p,\mathrm{s}_q}[i]$ and $U_{\mathrm{s}_p,\mathrm{s}_q}[i]$, given by the formulae:

$$\begin{aligned} L_{\mathrm{s}_p,\mathrm{s}_q}[i] &:= \min\{\, j \in \{l_q+1, \ldots, u_q\} \mid C_{\mathrm{s}_p,\mathrm{s}_q}[i,j] > C_{\mathrm{s}_p,\mathrm{s}_q}[i,j-1]\,\}, \\ U_{\mathrm{s}_p,\mathrm{s}_q}[i] &:= \max\{\, j \in \{l_p, \ldots, u_p-1\} \mid C_{\mathrm{s}_p,\mathrm{s}_q}[i,j] > C_{\mathrm{s}_p,\mathrm{s}_q}[i,j+1]\,\}, \end{aligned}$$

When determining the $q$-th slicing site $c_q$, $2 \leq q \leq k$, for already fixed sites $\hat{c}_1, \hat{c}_2, \ldots, \hat{c}_{q-1}$, any value of the $\hat{c}_p$-th row in the matrix $C_{\mathrm{s}_p,\mathrm{s}_q}$, $1 \leq p < q$, leading to a sum

$$C = \sum_{1\leq p<p'<q} \alpha_{p,p'} \cdot C_{\mathrm{s}_p,\mathrm{s}_{p'}}[\hat{c}_p, \hat{c}_{p'}] + \sum_{1\leq p<q} \alpha_{p,q} \cdot C_{\mathrm{s}_p,\mathrm{s}_q}[\hat{c}_p, c_q]$$

larger than $\widehat{C}$, with $c_q < \min_{1\leq p<q} L_{\mathrm{s}_p,\mathrm{s}_q}[\hat{c}_p]$, thus gives a sum larger than $\widehat{C}$ for every $c'_q \leq c_q$. A corresponding statement holds for $U$.

With this approach, we were able to speed up DCA by a factor of 2. For more than 8 sequences, the improvement was even better: it is possible to align up to twelve sufficiently related sequences (e.g. the homology family of a protein) in rather moderate time spans (often just a few seconds, sometimes some minutes).

An addition to be tested soon is to compute, for each row $i$ ($l_p \leq i \leq u_p$) in $C_{\mathrm{s}_p,\mathrm{s}_q}$ and each $j$ between $L_{\mathrm{s}_p,\mathrm{s}_q}[i]$ and $U_{\mathrm{s}_p,\mathrm{s}_q}[i]$ the values

$$l^{p,q}_{i,j} := \min\{\, C_{\mathrm{s}_p,\mathrm{s}_q}[i,j'] \mid l_q \leq j' \leq j\,\}$$

and

$$u_{i,j}^{p,q} := \min\{\, C_{s_p,s_q}[i,j'] \mid i \leq j' \leq u_q\}$$

and to stop going left (or right) with $c_q$ whenever an appropriate sum including $\alpha_{p,q} \cdot l_{i,j}^{p,q}$ (or $\alpha_{p,q} \cdot u_{i,j}^{p,q}$, respectively) rather than $\alpha_{p,q} \cdot C_{s_p,s_q}[i,j]$ exceeds the given bound. On the expense of some more storage requirements, it can be hoped that again a considerable speed up will be achieved this way.

## 3.6 Performance of DCA

We have tested DCA thoroughly, using families of related random sequences as well as real biological data. The following are the main results of the evaluation:

- The memory usage of DCA is in the magnitude required for standard pairwise alignments (about 30 megabytes for twelve sequences of average length 250 – and just 16 times that much for an average length 1000) (cf. [45],[41]).
- Compared to previous simultaneous alignment methods, the program is very fast (about 40 seconds for twelve sequences of average length 250) (cf. [42]).
- The alignments are of a very high quality, in mathematical as well as in biological terms. For none of the analyzed random sequence families for which the optimal score could be computed using MSA, the sum-of-pairs score of the alignment computed with DCA differs by more than 0.3 percent from the optimal score [41]. Applied to biological sequences, DCA can compete with the best alignment methods currently available (cf. [20],[36]).
- Due to the simultaneity, the computed alignments are also very well suited as an unbiased starting point for the reconstruction of evolutionary relationships (cf. [34]).
- Because DCA approximates the optimal score very closely, new ways of testing and validating alternative choices for multiple alignment score functions are possible.
- Due to the stable interdependence of the parameters of DCA and its performance, the behavior of the program is transparent to the user.

In conclusion, we have shown that – although the multiple sequence alignment problem has been a much studied subject over the last decades – the systematic application of the well-known Divide&Conquer principle opens the way to a new, efficient, and effective simultaneous multiple sequence alignment algorithm.

## Acknowledgement

This paper was written while the second and the third named author were hosted by the Biomathematics Research Centre at the Mathematics Department, University of Canterbury, New Zealand, which is gratefully acknowledged.

# References

[1] S.F. Altschul. Gap Costs for Multiple Sequence Alignment. *J. Theor. Biol.* 138, pages 297–309, 1989.

[2] S.F. Altschul, R.J. Carroll, and D.J. Lipman. Weights for Data Related by a Tree. *J. Mol. Biol.*, 207, pages 647–653, 1989.

[3] S.J. Austin, P.W. Fowler, P. Hansen, D.E. Manolopoulos, and M. Zheng. Fullerene Isomers of $C_{60}$, Kekule Counts versus Stability. *Chem. Phys. Letters*, 228:478–484, 1994.

[4] D. Babić, G. Brinkmann, and A. Dress. Topological Resonance Energy of Fullerenes. in preparation, 1997.

[5] G. Brinkmann. The Combinatorial Enumeration of Tube-Type Fullerenes and Fullerene Caps. in preparation, 1997.

[6] G. Brinkmann and A.W.M. Dress. A Constructive Enumeration of Fullerenes. To appear in *Journal of Algorithms*, 1997.

[7] H. Carrillo and D. Lipman. The Multiple Sequence Alignment Problem in Biology. *SIAM J. Appl. Math.*, 48(5), pages 1073–1082, 1988.

[8] S.C. Chan, A.K.C. Wong, and D.K.Y. Chiu. A Survey of Multiple Sequence Comparison Methods. *Bull. Math. Biol.*, 54(4), pages 563–598, 1992.

[9] H.S.M. Coxeter. *Regular Polytopes.* Dover Publications, Inc, 1973.

[10] M.O. Dayhoff, R.M. Schwartz, and B.C. Orcutt. A Model of Evolutionary Change in Proteins. In M.O. Dayhoff, editor, *Atlas of Protein Sequences and Structure*, volume 5, suppl. 3, pages 345–352. National Biomedical Research Foundation, Washington, D.C., 1979.

[11] R.F. Doolittle. Molecular Evolution: Computer Analysis of Protein and Nucleic Acid Sequences. *Methods in Enzymology* 183, Academic Press, San Diego, 1990.

[12] A.W.M. Dress and M. Krüger. Parsimonious Phylogenetic Trees in Metric Spaces and Simulated Annealing. *Adv. in Appl.Math.* 8, pages 8–37, 1987.

[13] A.W.M. Dress. On the Computational Complexity of Composite Systems. Proceedings of the IX. Sitges Conference in Theoret. Physics, Sitges 1986, Springer Lecture Notes 268, pages 377–388, 1987.

[14] A.W.M. Dress. Computing Spin-Glass Hamiltonians. Manuscript, Bielefeld, 1986.

[15] A.W.M Dress, G. Füllen, and S.W. Perrey. A Divide and Conquer Approach to Multiple Alignment. In *Proc. of the Third Conference on Intelligent Systems for Molecular Biology, ISMB 95*, pages 107–113. AAAI Press, Menlo Park, CA, USA, 1995.

[16] D.-F. Feng and R.F. Doolittle. Progressive Sequence Alignment as a Prerequisite to Correct Phylogenetic Trees. *J. Mol. Evol.* 21, pages 112-125, 1987.

[17] O. Gotoh. An Improved Algorithm for Matching Biological Sequences. *J. Mol. Biol.* 162, pages 705–708, 1981.

[18] S.K. Gupta, J.D. Kececioglu, and A.A. Schäffer. Improving the Practical Space and Time Efficiency of the Shortest-Paths Approach to Sum-of-Pairs Multiple Sequence Alignment. *J. Comp. Biol.* 2(3), pages 459–472, 1995.

[19] T. Harmuth. Die Generierung simpler, 3-regulärer planarer, zusammenhängender Graphen mit vorgegebenen Flächengrößen. *Diplomarbeit,* Universität Bielefeld, 1997.

[20] R.E. Hickson, C. Simon and S.W. Perrey. An Evaluation of Multiple Sequence Alignment Programs Using an rRNA Data Set. submitted, 1997.

[21] D.S. Hirschberg. A Linear Space Algorithm for Computing Maximal Common Subsequences. *Communications of the ACM,* 18(6), pages 341–343, 1975.

[22] D.J. Klein and X. Liu. Elemental Carbon Isomerism. In *International Journal of Quantum Chemistry,* Quantum Chemistry Symposium 28, pages 501–523. John Wiley & Sons, 1994.

[23] H.W. Kroto, J.R. Heath, S.C. O'Brien, R.F. Curl, and R.E. Smalley. $C_{60}$: Buckminsterfullerene. *Nature,* 318, pages 162–163, 1985.

[24] D.J. Lipman, S.F. Altschul, and J.D. Kececioglu. A Tool for Multiple Sequence Alignment. *Proc. Natl. Acad. Sci. USA*, 86, pages 4412–4415, 1989.

[25] X. Liu, D.J. Klein, T.G. Schmalz, and W.A. Seitz. Sixty-Atom Carbon Cages. *Journal of Computational Chemistry,* 12(10), pages 1265–1269, 1991.

[26] J. Malkevitch. Polytopal graphs. In L.W. Beineke and R.J. Wilson, editors, *Selected Topics in Graph Theory,* vol.3, pages 169–188. 1988.

[27] H.M. Martinez. A Flexible Multiple Sequence Alignment Program. *Nucl. Acids Res.* 16, pages 1683–1691, 1988.

[28] D.E. Manolopoulos and P.W. Fowler. *An Atlas of Fullerenes.* Oxford University Press, 1995.

[29] D.E. Manolopoulos, J.C. May, and S.E. Down. Theoretical Studies of the Fullerenes: $C_{34}$ to $C_{70}$. *Chemical Physics letters,* 181, No 2,3, pages 105–111, 1991.

[30] M.A. McClure, T.K. Vasi, and W.M. Fitch. Comparative Analysis of Multiple Protein-Sequence Alignment Methods. *J. Mol. Biol. Evol.*, 11(4), pages 571–592, 1994.

[31] E.W. Myers and W. Miller. Optimal Alignments in Linear Space. *CABIOS*, 4(1), pages 11–17, 1988.

[32] E.W. Myers. An Overview of Sequence Comparison Algorithms in Molecular Biology. Technical Report TR 91-29, University of Arizona, Tucson, Department of Computer Science, 1991.

[33] S.B. Needleman and C.D. Wunsch. A General Method Applicable to the Search for Similarities in the Amino Acid Sequence of Two Proteins. *J. Mol. Biol.*, 48, pages 443–453, 1970.

[34] S.W. Perrey, M.D. Hendy, and R.E. Hickson. Evaluating the Bias of Multiple Sequence Alignment Methods for Phylogenetic Tree Reconstruction. Manuscript, Christchurch, 1997.

[35] S.W. Perrey and J. Stoye. Fast Approximation to the NP-hard Problem of Multiple Sequence Alignment. Information and Mathematical Sciences Reports, Series B:96/06 (ISSN 1171-7637), May 1996.

[36] S.W. Perrey, J. Stoye, V. Moulton, and A.W.M. Dress. The Simultaneous Alignment of Sequences Using the Divide and Conquer Approach. submitted, 1997.

[37] Chih-Han Sah. Combinatorial Construction of Fullerene Structures. *Croatica Chemica Acta*, pages 1–12, 1993.

[38] D. Sankoff and J.B. Kruskal, editors. *Time Warps, String Edits, and Macromolecules: The Theory and Practice of Sequence Comparison.* Addison Wesley, Reading, Mass., 1983.

[39] T.G. Schmalz, W.A. Seitz, D.J. Klein, and G.E. Hite. Elemental Carbon cages. *J. Amer. Chem. Soc.*, 110, pages 1113–1127, 1988.

[40] J. Stoye. Divide and Conquer Multiple Sequence Alignment. `http://bibiserv.techfak.uni-bielefeld.de/dca/`, 1996.

[41] J. Stoye, S.W. Perrey, and A.W.M. Dress. Improving the Divide-and-Conquer Approach to Sum-of-Pairs Multiple Sequence Alignment. *Appl. Math. Lett.*, 1997. To appear.

[42] J. Stoye. Divide-and-Conquer Multiple Sequence Alignment. *Dissertation*, Technische Fakultät der Universität Bielefeld, 1997.

[43] W.R. Taylor. Identification of Protein Sequence Homology by Consensus Template Alignment. *J. Mol. Biol.* 188, pages 233–258

[44] J.D. Thompson, D.G. Higgins, and T.J. Gibson. CLUSTAL W: Improving the Sensitivity of Progressive Multiple Sequence Alignment through Sequence Weighting, Position-specific Gap Penalties and Weight Matrix Choice. *Nucl. Acids Res.*, 22, pages 4673–4680, 1994.

[45] U. Tönges, S.W. Perrey, J. Stoye, and A.W.M. Dress. A General Method for Fast Multiple Sequence Alignment. *Gene*, 172, pages GC33–GC41, 1996.

[46] M. Vingron and P. Argos. Motif Recognition and Alignment for Many Sequences by Comparison of Dotmatrices. *J. Mol. Biol.* 218, pages 33-43, 1991.

[47] R.A. Wagner and M.J. Fischer. The String-to-String Correction Problem. *Journal of the ACM*, 21(1), pages 168–173, 1974.

[48] M.S. Waterman. *Introduction to Computational Biology. Maps Sequences and Genomes.* Chapman & Hall, London, UK, 1995.