Solving the Protein Threading Problem in Parallel

Nicola Yanev*
University of Sofia
5,J. Bouchier str.1126 Sofia, Bulgaria
choby@math.bas.bg

Rumen Andonov†
IRISA, Campus de Beaulieu
35042 Rennes Cedex, France
randonov@irisa.fr

Abstract

We propose a network flow formulation for protein threading and show its equivalence with the shortest path problem on a graph with a very particular structure. The underying Mixed Integer Programming (MIP) model proves to be very appropriate for the protein threading problem–huge real-life instances have been solved in a reasonable time by using only a Mixed Integer Optimizer instead of a special-purpose branch&bound algorithm. The properties of the MIP model allow decomposition of the main problem on a large number of subproblems (tasks). We show in this paper that a branch&bound alike algorithm can be efficiently applied to solving in parallel these tasks, which leads to a significant reduction in the total running time. Computational experiments with huge problem instances are presented.

1. Introduction

The *protein threading* problem is an extremely important challenge in computational biology [2, 6, 9, 10]. The problem consists of testing whether or not a target sequence *query* is likely to fold into a 3D template structure *core* by searching for an *alignment* which minimizes a suitable *score function*. It is important, because the biological function of proteins is determined by their three-dimensional shape, and their shape is determined by their linear sequence.

We give a more formal presentation of the problem by simultaneously introducing a preexisting terminology. Let the core contain a set of m items S_i, called *segments*, each of length l_i. This set must be *aligned* to a sequence L of N characters from some finite alphabet. Let t_i be the position in L where S_i starts. An alignment is called a *feasible threading* if:

i. $t_i \geq t_{i-1} + l_{i-1}$ for all i;

ii. the length g_i (called *gap* or *loop*) of uncovered characters, i. e. $g_i = t_i - t_{i-1} - l_{i-1}$ is bounded, say $g_i^{min} \leq g_i \leq g_i^{max}$.

The formal model of the protein threading problem accepted by us in this paper is very close to the one given in [1, 2]. For given integers N, m and $l_i, i = 1, \ldots, m$, let us define $n = N + 1 - \sum_{i=1}^{m} l_i$. Let the set of couples $R \subset \{(i,j) \mid 0 < i < j \leq m\}$ and the reals $c_{ij}, i = 1, \ldots, m, j = 1, \ldots, n$, and $c_{ikjl}, (i,j) \in R, 1 \leq k \leq l \leq n$, be given. The goal is to minimize

$$\sum_{i=1}^{m} c_{ik_i} + \sum_{(i,j)\in R} c_{ik_ijk_j} \qquad (1)$$

subject to: $1 \leq t_1 \leq n$; $t_i \geq t_{i-1} + l_{i-1}, i = 2, \ldots, m$; $t_m + l_m \leq N + 1$; $k_1 = t_1$; $k_i = t_i - (t_{i-1} + l_{i-1} - 1), i = 2, \ldots, m$; and $t_i, i = 1, \ldots, m$, are integers. In this definition n corresponds to the "degree of freedom" (the number of relative positions) for each segment; R is associated with the set of pairwise interactions between the segments; the coefficients c_{ikjl} score for a placement of S_i at the k^{th} relative position and of S_j at the l^{th} one; c_{ij} score the placement of the i^{th} segments at the j^{th} relative position.

Finding a really fast algorithm for solving this optimization problem it is still considered as a challenge. The problem has been proven to be NP-complete in [4] and to be MAX-SNP-hard (which means that it cannot be approximated to arbitrary accuracy in polynomial time) in [1]. The main inspiration for us to start working over this subject is the results announced in [2, 3] describing a branch-and-bound algorithm (b&b) which is successful for a large set of real-life (biological) ex-

*The work of N. Yanev has been partially supported by the GenoGRID project (ACI GRID, Ministère de la Recherche) and was performed during a visit to the SYMBIOSE project, IRISA, Rennes

†On leave from LAMIH/ROI, University of Valenciennes, France, Rumen.Andonov@univ-valenciennes.fr, corresponding author.

amples. We were aware of the fact that the real-life instances may be more tractable than randomly generated instances. Nevertheless, a branch-and-bound algorithm solving a non-linear integer programming problem over the search space of size up to 10^{31} feasible threadings by using relaxation of non-evident quality is reason enough for asking questions like: i) how intrinsically hard is the problem on real biological instances (especially for so called self threading)?; ii) could it be improved (either by improving bounds or by creating another mathematical programming model)?. In addition we may also answer; iii) can we use the power of parallel computers in order to speed up the computation?; iv) is it necessary to write special purpose code or could some existing solvers be used? The answers to these questions are given in the subsequent sections.

The main contributions of this paper are as follows

- Protein threading is formulated as a network flow model and has proved to be equivalent to the shortest path problem (SPP) on a graph with a particular structure. All previous results use algorithms like branch-and-bound algorithm [2, 3], or divide-and-conquer [6], but to the best of our knowledge, such a mathematical programming approach has been never proposed for this problem before.
- Three MIP models are proposed, analyzed and compared to each either, and with the popular Lathrop_Smith's (b&b) algorithm [3]. In all cases the MIP formulation outperforms significantly the popular (b&b) algorithm.
- The MIP polytope posseses an extremely useful property: in all but very few cases the solution is given by solving the LP relaxed problem. In the rare exceptions the integer solution is easily found by the standard CPLEX heuristic (i.e. also in a polynomial time).
- A divide and conquer strategy is proposed for the formulated SPP problem. This strategy leads to a significant reduction of the computation time.
- Several parallel schemes have been implemented and analysed in order to improve their efficiency. Extensive computational experiments have been performed for huge problem instances.

This paper is based on results described in details in a previous research report [11]. At the time of its publication, we were aware of no any other MIP approach for the protein threading problem. We are pleased to observe that recentlty, another research team presented similar approach [13]. Since analytical comparison is very difficult, it will be worthwhile to work toward a creation of a common benchmark, in order to be able to compare different algorithmic approaches.

2. Network flow formulation

Let $G(V,E)$ be a digraph with vertex set $V = \{(i,k) \mid i = 1,m;\ \ k = 1,n\}$ and arc set $E = \{((i,k),(i+1,l)) \mid i = 1,m-1;\ \ l \geq k\}$, and let $NL = \{(i_1,j_1),...,(i_t,j_t)\}$ be the set of non-local $(i_s < j_s - 1)$ interactions. This induces the set of indices $E^{ind} = \{(i_k l j_k f) \mid f \geq l, k = 1,t;\ \ l = 1,n\}$ meaning that the left segment i_k of the non-local pair $(i_k,, j_k)$ is placed at position l and the right segment j_k of the same pair at position f. Each layer $(i,k), k = 1,n$ corresponds to the possible placements of the i^{th} segment relatively to the end of $(i-1)^{th}$ segment. By adding two extra vertices S and T and arcs $(S,(1,k)), k = 1,n$ and $((m,k),T), k = 1,n$, one could easily check 1 to 1 correspondence of the previously defined threadings with the (S,T) paths in G. The condition $l \geq k$ prevents overlapping of any two consecutive segments and if the path passes through vertices $(i-1,k),(i,l)$ then $l-k$ is the size of the gap g_i. We introduce also variables x_e for $e \in E$ and z_e for $e \in E^{ind}$. Now we can model the problem in the following way:

$$\sum_{e \in E} c_e x_e + \sum_{e \in E^{ind}} c_e z_e \Rightarrow \min \tag{2}$$

such that:

$$\sum_{e \in \Gamma(i,k)} x_e - \sum_{e \in \Gamma^{-1}(i,k)} x_e = 0 \quad \forall (i,k) \in V \tag{3}$$

$$\sum_{e \in \Gamma(S)} x_e = 1 \tag{4}$$

$$z_c \leq \sum_{e \in \Gamma(i_k,l)} x_e,\ \forall c = (i_k,l,j_k,f) \in E^{ind} \tag{5}$$

$$z_c \leq \sum_{e \in \Gamma^{-1}(j_k,f)} x_e,\ \forall c = (i_k,l,j_k,f) \in E^{ind} \tag{6}$$

$$\sum_{e \in \Gamma(i_k,l)} x_{e+} \sum_{e \in \Gamma^{-1}(j_k,f)} x_e - z_c \leq 1 \quad \forall c \in E^{ind} \tag{7}$$

$$\sum_{f \geq l} z_{i_k l j_k f} = 1 \quad \forall (i_k,j_k) \in NL \tag{8}$$

$$z - \text{ binary}, x \geq 0 \tag{9}$$

Into this model, $\Gamma(x)$ is the set of arcs outgoing from vertex x and $\Gamma^{-1}(x)$ is the set of in-going arcs. Constraints (3), (4) are network flow representation of the paths from S to T (see fig.1), constraints (5) and (6) force the path to pass through the pair of vertices activated by SOS (Special Ordered Set) constraints (8), constraints (7) are for tightening the LP-relaxation. This model was built under the presumption to enumerate the smaller search space of variables z than of x, that will be the case if we switch the integer requirements from z to x variables (this need some minor changes in the model). From the well known properties of the network flow polytope one can see that for each fixation of z variables to $0, 1$ the respective vertices of the underlying polytope in $(x, z)-$space have $(0, 1)$ x coordinates. To conclude we need to say how the arcs weights c_e are related to the scores from the introduction. Each weight is a sum of three numbers: one for the tail of the arc (segment-to-position cost), second for the gap cost between segments(if any) and the third is for the local interactions (if any). If the leading and/or trailing gaps are scored then these numbers are prescribed to the out-going/in-going arcs from/to the vertices on the first/last layer. From the graph in fig.1 we could have more geometrical insight for the problem of optimal aligning of some sequence with a core of five segments, each one with three possible placements depicted as columns. The path given in thick lines has a length 5 but taking into account the pairwise interactions (in this case (1,1,3,2), (3,2,5,2) - the path passes through the vertices (1,1),(3,2), (5,2) we must add the costs for passing through these vertices ($c_{1132} + c_{3252}$) to 5 and to obtain the actual length 14 of the path (threading). Thus if we have given weights to all arcs and a table of the costs for the designated non-local pairwise interactions the optimization problem will convert to finding a path from S to T with minimal updated length. From this figure we can also stress z- to- x relation: once the z variables are fixed ($z_{1132} = 1, z_{3252} = 1$, all other are zero because of (8)) to find which x will be fixed to 1 is equivalent to find the shortest from among the paths passing through (1,1), (3,2), (5,2).

3. The self-threading case

In [2, 3] a branch&bound algorithm (later referred to as **LS**) is given together with the demonstration of its effectiveness on a wide range of instances. A list of impressive computational results is given on a reach set of s.c. self-threading (the protein sequence is aligned with its own core) instances. When we run CPLEX on the MIP models generated according to (2)-(9) on a large subset of these instances the results are always: *the LP relaxation attains its minimum at a feasible (0,1) vertex, hence optimal*. This property is so pertinent to the model that one could use it for defining the self-threading subclass. The relaxed problem used in **LS** model [2, 3] is based on minimizing of a function $\underline{f}(x)$ which is inferior to the objective function $f(x)$ over the set of feasible threadings. For such a relaxation, an optimal solution x^* to the relaxed problem is optimal for the original one if $f(x^*) = \underline{f}(x^*)$. For the **LS** model this could be taken as the self-threading subclass defining property and this is the reason for the effectiveness of the **LS** algorithm on the instances of this subclass. What is important to add here is that all these properties are score depended and they could be lost once the scoring scheme changed. For all instances reported below the objective function coefficients are generated by using FROST (*Fold Recognition Oriented Search Tool*) software [7, 8].

4. Further improvements of the MIP model

In order to improve the LP-bounds and the CPLEX branching strategy by imposing branching on the SOS constraints instead of on a single variable (but at the expense of adding extra constraints) the following modification of the model (2)-(8) is done:

Let $L(NL) = \{i_s|(i_s, j_s) \in NL\}$, $R(NL) = \{i_s|(j_s, i_s) \in NL\}$.

$$\begin{cases} y_{il} = \sum_{f \geq l} z_{iljf}, l = 1, \ldots, n; \\ \qquad\qquad \forall i \in L(NL)\ (i,j) \in NL \end{cases} \tag{10}$$

$$\begin{cases} y_{il} = \sum_{f \leq l} z_{jfil}, l = 1, \ldots, n; \\ \qquad\qquad \forall i \in R(NL)\ \ (j,i) \in NL \end{cases} \tag{11}$$

$$\sum_{l=1}^{n} y_{il} = 1 \quad \forall i \in R(NL) \bigcup L(NL) \tag{12}$$

$y-$binary and all z variables included into these definitions change from binary to continuous. The role of the newly introduced constraints is twofold: i) (10)-(12) are tighter than (8) thus improving LP-bounds and could be used for much more flexible branching strategy through explicitly introducing them in the SOS section of the model to be solved by CPLEX ; ii) the number of the (0,1) variables is drastically reduced because the z variables, covered by the defining constraints, are forced to be integer in consequence of the unimodularity

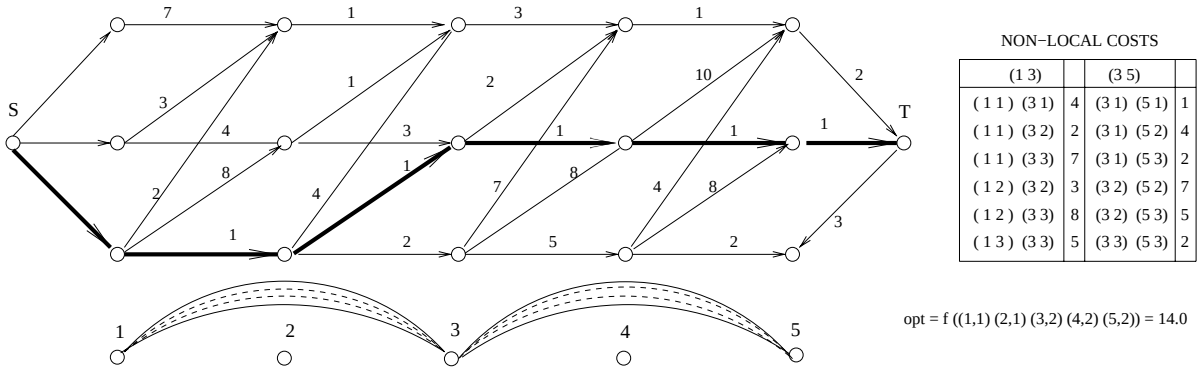

Figure 1. Graph corresponding to the network flow formulation of the problem. $f(x)$ is a function computing the exact cost of the path x.

of the corresponding matrix. Although, methodologically this model (denoted here by **M2**) is very important for our approach its main drawback is the size of the problems created (see [11]). For instance, when aligning the protein 1COY_0 with the core 1GAL_0 (a problem of size 36 segments and 81 positions) we observe that the corresponding MIP problem has 741 264 rows 360 945 columns and 54 145 231 nonzero elements. This is, of course, prohibitively large for practical use and appeal for splitting techniques (see section 6) which help for significant reduction of the MIP problem size and also of the solution time. One could feel that what impacts mainly on the size of the optimization problem (besides m, n) is the two level control adopted: x variables are controlled by z variables which are controlled by y variables. How to overcome this deficiency of the model is described in the next section.

5. The main result

Now we introduce binary variables $y_{ij},\ i \in L(NL) \cup R(NL)\ j = 1, ..., n$ to prevent the flow passing through vertex (i, j) when $y_{ij} = 0$ or direct through it when $y_{ij} = 1$. This could be modeled in an obvious way to obtain the following network-flow alike problem (**M***):

$$\sum_{i=1}^{m-1} c_i x_i + \sum_{i \in NL} c_i z_i \Rightarrow \min \tag{13}$$

subject to

$$A_i z_i - I y_i = 0 \ \ i \in L(NL) \cup R(NL) \tag{14}$$

$$B_i x_i - I y_i = 0 \ \ i \in L(NL) \cup R(NL) \tag{15}$$

$$Fx = l \tag{16}$$

$$Ey = 1 \tag{17}$$

$$y_{ij} \in \{0, 1\} \tag{18}$$

where A_i are node-arc incidence matrix for the nodes $(i, 1), ..(i, n)$ and the z-arcs out-going/in-going from/to these nodes, B_i are node-arc incidence matrices for the same nodes but for the $x-$arcs, F is the node -arc incidence matrix for the digraph G, thus (16) are the classical network flow constraints, l is zero vector except for the first n components equal to y_{1j} and finally (17) which are SOS constraints on (0,1) variables y. All matrices are absolutely unimodular and the problem is of block angular form with y as the only binding variables.

Starting tests with this model on various biological examples, surprisingly the *LP solution happened to be integer*, i.e. path/threading defining, for a large set of instances. Table 2 contains results obtained when aligning the biggest queries in FROST data base. What is seen is that even for polytopes of at least 10^{39}vertices, the objective function of the relaxed problem of (13)-(18), counting more than million variables, attains its minimum at a threading defining vertex. Very few instances (in fact only four till now) have been found where the solution of the LP relaxed problem is not integer. As we see from table 3, for all these cases CPLEX succeeds to find the integer solution in a short time after solving the LP relaxed problem–and this happens prior to the branching phase and using local search heuristic only. Moreover, the LP relaxation/integer solution gap is so

small (see table 3) that the LP objective function value could perfectly guess the closest core in a single query-to-multiple cores mode. One could see a lot of similarities with the classical incapacitated plant-location problem, where the generating of difficult instances requires special efforts.

We therefore observe that when the set of query-to-core instances is restricted to real ones and the score function is as in [7, 8] then an algorithm exists which can solve each such problem instance in polynomial time.

We show in this way that only by using the M* model and the non-special purpose LP solver of CPLEX one could solve in an affordable time all practical problems in the context of protein threading. (The problems with non-feasible LP solution could be driven to LP feasibility by using an ad-hoc procedure instead of standard CPLEX heuristics, but this will be worthwhile to implement only if such "bad" instances persist).

Some hints for further improvement of the effectiveness of the approach is discussed in section 6. As for the minimalilty of polytope describing constraints, we must note that each attempt of aggregation, say in (14), spoils the feasibility of the LP solution.

6. Split and conquer

Let us recall here that by the very definition of (13)-(17) model, one can partition (split) it to smaller subproblems by imposing constraints on the paths in the graph G in the following way. Let k be some segment and (k, j) $j = 1, ...n$ are the vertices in the k^{th} layer. Then by partitioning the set $\{1, 2, ..., n\}$ into r intervals, we could split the problem into r subproblems of smaller sizes and the i^{th} one is defined over the paths in G which pass through the vertices in the i^{th} interval only. Thus if this interval is (p, q) then for the segments l on the left(right) of k only the vertices $(l, s) : s \leq q$ $((l, s) : s \geq p)$ should be taken into account. The best choice for k is $\lceil 0.5m \rceil$ and if the goal is to split the problem into subproblems of approximately equal size then the intervals should be of equal length. One could target splitting on the criterium - almost equal number of paths, but because of simplicity of the implementation we used the first approach. Now, if the split is done one can start solving the subproblems in some order by passing the best objective function value found as a cutoff for the subsequent subproblems. Thus, by having the chance to start with the subproblem which contains the optimal path, all other subproblems will be aborted by the LP solver at the moment when the dual objective reaches the cutoff value. The effect of this strategy is demonstrated in table 4 on instances which are worthwhile to split. One good choice for the problems to start with is based on the observation that for the biological instances the optimal path passes "near" the middle vertex in the middle layer (this area is a crossroad of maximal number of paths). The estimate of this nearness could be given as a parameter for sequential or as well for parallel implementation of such splitting approach.

7. Parallelization

The splitting strategy from the previous section suggests the following "naive" parallelization. The r subproblems can be considered as tasks which need to be spread over p processors. The solution of the original problem is the minimum of all solutions so computed. To obtain an efficient code we also need to implement the "learning" effect – the best objective function value is passed as cutoff from the previous to the subsequent subproblems. The parallel algorithm we propose is based on *centralized dynamic load balancing*: tasks are handed out from a centralized location (pool) in a dynamic way. The work pool is managed by a "master" which gives work *on demand* to idle "slaves" and also passes them the best objective value found from the previous tasks. Each slave applies (**M***) model to solve the corresponding subproblem. Note that the dynamic load balancing is the only reasonable one in the context of irregular tasks where the amount of work is not known prior to execution.

In our first parallel algorithm the tasks were atomic i.e. without communication during task execution. This implementation was, however, frequently slower than the sequential splitting strategy. In order to explain this phenomenon let us enumerate the following important properties we observed.

- The subproblems have a tendency to lose the "nice" property of the original problem and frequently the LP relaxation yields a non-integer solution. Such subproblems become "hard" problems.
- The lower bounds for these hard subproblems are such that the corresponding nodes in the b&b tree can be pruned by the value dominance condition.
- In the big majority of instances (in fact we found only one exception to this rule), the global solution has been obtained in an "easy" subproblem (i.e. when the LP relaxation was integer).

Taking this into account, we can better understand the observed behavior. There is a high probability in a

parallel implementation that a hard problem with a weak bound is among the first tasks to be computed. When the tasks are atomic, there is no learning for these first tasks before their end, and the hardest one can seriously slow down the global optimization process. The load balance is very bad. Often the "unlucky" processor (the one starting with a hard subproblem) resolves only one task. The global time is in fact determined by the slowest process.

We succeed in overcoming this weakness by making the tasks non-atomic, by using the CPLEX call-back-function technique [12]. This technique permits the user to perform some user defined operations either at any node of the tree search (in the case of Mixed Integer Optimizer) or at a fixed number of simplex iterations (in the case of LP solver)[1]. The operations we perform are : i) sending to the master the best integer value locally computed; ii) receiving from the master the current global record; iii) using it to update the cut-off value. This periodical updating of the local record with the best global value allows, the parallel processes to evolve much faster simultaneously. The communications are more frequent in this version, but since we deal with huge size instances, the cost of these additional communications is small compared to the tasks' granularity. Furthermore, this non-promising tasks cancellation, in due course leads to significant gain in the total time. Such "timely best value exchange" makes the algorithm robust, in the sense that the optimization speed is determined by the fastest (most successful) of all parallel processes.

8. Computational experiments

In tables 1 and 2, we summarize the results concerning the sequential model. They are obtained by running ILOG CPLEX Callable Library on SUNW, UltraSPARC-II, 400 MHz, CPU computer. The instances (scores) are drawn from a redirected output of FROST [7, 8] which tries to find the best fit of multiple queries-to-multiple cores bank. The LS algorithm [3] is used at a final stage of this complex and time-consuming procedure. In order to generate interesting (very big) instances we had to limit the time for this b&b code to an upper bound, varying between 30min. and 2h. according to the instance size. What we mean by interesting instance is one with more than 10^{31} feasible threadings. When LS reaches the associated time bound (respectively indicated by the sign • in the b&b "time" column of table 1) we write down the best value ever found by b&b. We see from table 1 that M* significantly outperforms the b&b algorithm. Table 2 gives results on instances never attempted before.

Table 4 illustrates the impact of the split and conquer strategy on the total running time, while table 5 presents experiments from its parallelization. We use MPI communication library, and the results are obtained on two of the above mentioned machines, each one of four processors. The gain of the parallelization clearly increases with the problem size and the results are very encouraging for this (relatively early) stage of development of our code. Current open questions are : what are the optimal values for the number of subproblems and for the number of processes. At that moment we use LP format file to instantiate CPLEX problem objects, and therefore the input/output operations take, today, a significant part of the total running time. This part even increases, when the number of subproblems augments, and this complicates the problem decomposition. Using the alternative way offered by CPLEX to populate problem objects will be our next step. With this next version, we plan to perform a detailed analysis, in order to tune the parameters influencing the program efficiency, such as: i) the number of simplex iterations between consecutive communication exchanges; ii) the number of generated subproblems; iii) the number of involved processes.

9. Conclusions

We have demonstrated, once more, that a marriage of mathematical programming with parallel programming theory and algorithms can be a valuable tool for attacking optimization problems now arising in computational biology. We succeed, relying on such achievements, to linearly model a problem of nonlinear combinatorial nature, and to efficiently solve a lot of instances without having written a single line of code (different from a model builder). This model reveals an unexpected property of protein threading problem, when it is considered only over biological instances, namely feasibility of the linear programming solutions. We also propose a splitting strategy, which is very convenient for parallel implementation. The preliminary results of our parallel code are extremely encouraging - huge problem instances have been solved in a very reasonable time.

10. Acknowledgement

We are grateful to Jean-François Gibrat and Antoine Marin for introducing the problem to us, for many help-

[1] fixed to 500 iterations for the presented results

query name	query size	core name	problem size segm.	problem size pos.	space size	B&B score	B&B time in sec.	M* score	M* time in sec.
2CYP_0	294	2CYP_0	15	98	1.5e+18	-1898.2	105	-1898.2	18
3GRS_0	478	3GRS_0	30	114	6.5e+30	-3809.7	446	-3809.7	68
1COY_0	507	1COY_0	27	149	4.0e+31	-3386.2	1247	-3386.2	108
3MINA0	491	3MINA0	33	116	1.1e+33	-3556.2	560	-3556.2	70
1GAL_0	583	1GAL_0	36	157	1.3e+39	-4042.2	1875	-4042.2	117
2CYP_0	294	1THEA0	13	138	1.8e+18	-11.4	• 1200	-11.6	606
3MINA0	491	3MINB0	33	62	2.5e+25	398.4	• 6074	390.1	361
1COY_0	507	1GAL_0	36	81	1.3e+30	100.0	• 1800	98.7	460
3MINA0	491	4KBPA0	23	189	3.2e+30	57.42	• 6469	57.42	3211
3MINB0	522	1GPL_0	23	215	5.3e+31	120.4	• 3000	63.5	2794
1GAL_0	583	1YVEI0	31	140	9.2e+33	66.19	• 42425	52.76	3827
1GAL_0	583	1COY_0	27	225	1.3e+36	-295.60	• 42600	-296.60	12061

Table 1. Huge instances: B&B versus M* comparison. Note that even for selfthreading (the first five instances) M* model is much faster than B&B. The sign • indicates that B&B has finished because of the time limit – the solution obtained in this case is not proven to be optimal and can be considered as an approximation.

query name	query size	core name	size segm.	size pos.	space size	LP size rows	LP size columns	iter	time in sec.	score
1GAL_0	583	1FIEB0	42	29	2.8e+19	6708	53127	9279	32	1023.03
3MINA0	491	3PMGA0	40	36	2.9e+21	8159	78464	13635	80	76.79
3MINA0	491	3MINB0	33	62	2.5e+25	11833	191259	23173	361	390.14
3MINB0	522	2MPRA0	20	161	1.8e+26	13477	550875	17764	698	84.53
3MINA0	491	1AOZA0	35	66	1.1e+27	16575	282818	39775	1074	405.66
3MINB0	522	5EAS_0	22	183	1.8e+29	19148	896063	15586	825	149.77
3MINA0	491	1BIF_0	25	150	1.1e+30	18815	728142	26980	1998	81.78
3MINA0	491	1INP_0	21	227	1.4e+30	23085	1349881	49217	8302	7.50
3MINA0	491	2GPL_0	23	184	1.8e+30	20539	957260	24506	1866	• 98.06
3MINA0	491	4KBPA0	23	189	3.2e+30	29371	1494701	34427	3131	57.42
3MINB0	522	1GPL_0	23	215	5.3e+31	24011	1305173	29351	2451	63.55
3MINA0	491	1PBGA0	32	123	1.2e+33	23479	743245	47746	3126	• 90.78
3MINB0	522	2YHX_0	26	218	6.6e+34	28907	1604695	54134	8842	• -11.82
1GAL_0	583	1COY_0	27	225	1.3e+36	36339	2065362	57045	12061	• -296.60
2CYP_0	294	3GRS_0	30	219	4.1e+38	41477	2294957	41782	3260	-230.44
1GAL_0	583	1AD3A0	31	212	1.3e+39	37195	1993288	100883	26018	76.28

Table 2. Huge instances solved by M*. The sign • marks cases when the LP relaxation is non-integer.

query name	core name	space size	LP solution	integer solution	time for LP relaxation	total time in sec.
3MINA0	2GPL_0	1.8e+30	97.43	98.06	1610	1866
3MINA0	1PBGA0	1.2e+33	90.23	90.78	2898	3125
3MINB0	2YHX_0	6.6e+34	-12.43	-11.82	7914	8841
1GAL_0	1COY_0	1.3e+36	-297.47	-296.60	11087	12060

Table 3. Comparing LP relaxation versus integer solution value/time

N^0	query name	core name	problem size		space size	(number_of_sub_problems : time)		
			segm.	pos.				
1	2CYP_0	1THEA0	13	138	1.8e+18	(1 : 10m 06s)	(5 : 4m 00s)	(3 : 3m 38s)
2	2BMHA0	1CEM_0	21	203	1.5e+29	(1 : 34m 45s)	(3 : 21m 12s)	(5 : 17m 54s)
3	3MINB0	1GPL_0	23	215	5.3e+31	(1 : 46m 34s)	(10 : 29m 30s)	(5 : 22m 03s)
4	2CYP_0	3GRS_0	30	219	4.1e+38	(1 : 58m 29s)	(9 : 49m 24s)	(5 : 32m 43s)
5	1GAL_0	1AD3A0	31	212	1.3e+39	(1 : 7h 10m 15s)	(13 : 3h 10m 53)	(9 : 1h 14m 23s)

Table 4. Huge instances: impact of split and conquer strategy in M* model

N^0	(number_of_processus : number_of_sub_problems : time)			
1	(1 : 3 : 3m 38s)	(6 : 13 : 2m 38s)	(4 : 8 : 2m 26s)	(5 : 3 : 2m 07s)
2	(1 : 5 : 17m 54s)	(5 : 5 : 12m 20s)	(4 : 7 : 12m 10s)	(4 : 15 : 11m 38s)
3	(1 : 5 : 22m 03s)	(4 : 5 : 13m 20s)	(5 : 7 : 12m 17s)	(7 : 9 : 10m 00s)
4	(1 : 5 : 32m 43s)	(3 : 5 : 32m 15s)	(5 : 9 : 34m 14s)	(5 : 15 : 30m 30s)
5	(1 : 9 : 1h 14m 23s)	(3 : 9 : 58m 44s)	(5 : 13 : 43m 12s)	(7 : 29 : 39m 50s)

Table 5. Splitting strategy parallel implementation. Instances numbers correspond to table 4

.

ful discussions, and for providing us with the code of Lathrop&Smith algorithm, as well as all data concerning the protein structure prediction problem. Special thanks are due to Stefan Balev, who participated actively in the initial stage of this study.

References

[1] T. Akutsu and S. Miyano, On the approximation of protein threading, Theoretical Computer Science, 210 (1999), 261-275.

[2] R. Lathrop, R. Rogers Jr., J. Bienkowska, B. Bryant, L. Butorovic, C. Gaitatzes, R. Nambudripad, J. White, T. Smith, Analysis and Algorithms for Protein Sequence-Structure Alignment, Comp Methods in Molecular Biology, chapter 12, pp. 227-283, 1998.

[3] R. Lathrop, T. Smith, Global Optimum Threading with Gapped Alignment and Empirical pair Score functions, J. Mol. Biol., 255, 641-665, 1996.

[4] R. Lathrop, The protein threading problem with sequence amino acid interaction preferences is NP-complete, Protein Eng. 7, 1059-1068, 1994.

[5] F. Plastria, Formulating logical implications in combinatorial optimization, EJOR 140, 338-353, 2002.

[6] Xu Y., Xu D., Uberbacher E., An Efficient Computational Method for Globally Optimal Threading, J. Comp. Biol., V5, Number 3, 1998.

[7] A. Marin, J.Pothier, K. Zimmermann, J-F. Gibrat, Protein structure prediction: bioinfromatic approach, I. Tsigelny Ed. International University Line, 2002, chapter "Protein threading statistics: an attempt to assess the significance of a fold assignment to a sequence"

[8] A. Marin, J.Pothier, K. Zimmermann, J-F. Gibrat, FROST: A Filter Based Recognition Method, to appear in Proteins: Struct. Funct. Genet., vol. 49, 2002

[9] J.C. Setubal, J. Meidanis, Introduction to computational molecular biology, 1997, International Thomson Publishing Inc.

[10] T. Lengauer, Computational Biology at the Beginning of the Post-genomic Era, LNCS, Vol. 2000, "Informatics: 10 Years Back - 10 Years Ahead", R. Wilhelm(Ed.), Springler, Berlin 2000, P. 341-355

[11] N. Yanev and R. Andonov, The Protein Threading Problem is in P?, RR INRIA, No 4577, October 2002 (http://www.inria.fr/rrrt/rr-4577.html)

[12] ILOG CPLEX 7.0 reference manual, www.ilog.com

[13] J. Xu, M. Li, G. Lin, D. Kim and Y. Xu, Protein threading by linear programming, PSB, 2003, January, 2003