# Entwurf fuer ein Paper, voerlaeufiger Titel: Sidechain placing in Homology Modeling via Lagrangian Relaxation, (zu verwenden fuer die Promotion)

Samir Mourad

July 21, 2005

# Sidechain placing in Homology Modeling via Lagrangian Relaxation

Samir Mourad[1] and Oliver Kohlbacher[2]

February 2004/Mrz-Mai 2005

**Abstract**

We illustrate a new approach to the sidechain placing problem. The approach is based on formulating the problem as an integer linear program and then relaxing in a Lagrangian way a suitable set of constraints.

*Key Words:* molecular modeling, discrete optimization, Lagrangian Relaxation

## 1 Introduction

### 1.1 Sidechain placing in homology modeling

Conformatitions occuring in proteins can be adequately described by a rather small set of so-called rotamers for each amino acid. These rotamer libraries can be used to reduce the sidechain placement problem to a combinatorial optimization problem: search for the set of rotamers with the minimum energy, i.e., the global minimum energy conformation (GMEC). As the number of rotamer combinations is very high (...), efficient methods are required to identify the GMEC or suboptimal solutions sufficiently close to the GMEC.

### 1.2 Sidechain conformation optimization

In [3] a combinatorial approach for sidechain conformation optimization in Protein Docking area is introduced. There are introduced two methods. One uses an integer linear program and branch-and-cut algorithm. In [8] the constraints of the integer program of [3] are improved.

In this paper a Lagrangian Relaxation (LR) approach is introduced for sidechain conformation optimization to be used in sidechain placing for homology modeling of proteins. The theory of Lagrangian Optimization is a well established branch in of Combinatorial Optimization and has been used sucessfully in a large number of applications, in different domains [2]. Recently [5] decribed an LR approach for Structural Alignment of Large-Size Proteins this was the first time that a similar approach was used for an alignment problem in Computational Molecular Biology. Nowadays, LR is the most successful tool to tackle very large problems. These algorithms are capable of finding near-optimal solutions to instances with millions of variables and thousands of constraints within minutes on a PC.

[1]Universitaet Tuebingen, Wilhelm Schickard Institute for Computer Science, Dept. for Simulation of biological Systems, Sand 14, D-72076 Tuebingen and VGOEG, Haid-und-Neu-Str.7, D-76139 Karlsruhe, email: mourad@zgoeg.de

[2]Universitaet Tuebingen, Wilhelm-Schickard-Institute for Computer Science, Dept. for Simulation of biological Systems, Room C318, Sand 14, D-72076 Tuebingen

## 1.3 Lagrangian Relaxation

The LR approach is particularly well suited for those cases in which the formulation of a problem consists of two sets of constraints: a set of nice constraints and a set of bad constraints, whose removal makes the resulting problem, called the Lagrangian relased problem, easily solvable.

1.The strategy then consists in removing the bad constraints from the formulation and putting them into the objective function, each weighted by some coefficient (Lagrangian Multiplier). The weight for a constraint represents a penalty which is incurred by a solution which does not satisfy that constraint. To any choice of weights corresponds a (relatively easy) problem whose solution yields a bound to the original problem.

2.The core question of LR is then to determine the optimal weigths, i.e., the Lagrangian multipliers yielding the best bound. In most cases, the determination of these multipliers is equivalent to solving a suitable LP, which would be too time consuming in practice. On the other hand near-optimal multipliers can be found by a simple iterative procedure called subgradient optimization, in which, at each iteration, the Lagrangian relaxed problem is solved and the multipliers are updated based on the corresponding solution.

3.Besides yielding an upper bound on the optimal solution of the original problem, the Lagrangian multipliers (and the associated costs/profits in the objective function) can be used to drive simple heuristic procedures (in most cases of greedy nature). These procedures typically produce substantially different solutions for different Lagrangian multipliers.

4.Accordingly, if the Lagrangian multipliers are embedded within an iterative procedure to define near-optimal near-optimal multipliers, namely they are called at each iteration with the current multipliers, the best solution found over all iterations tends to be near-optimal.

### 1.3.1 Design of a general LR/MIP algorithm

The following introduction to LR is from [6].

Lagrangean Relaxation is a Price Directive decomposition technique, which in the first instance simplifies and reduces the problem in question by relaxing groups of constraints. Lagrangean relaxation has been successfully used in processing many different instances of combinatorial optimisation problems, such as the Travelling salesman Problem. Many combinatorial optimisation problems consist of an easy problem that is complicated by the addition of extra constraints. Applying LR in these problems involves identifying these complicating constraints, and then relaxing them by attaching penalties to the complicating constraints and then absorbing them into the objective function. These penalties are known as the Lagrange multipliers. Due to the relaxation of the complicating constraints, the relaxed problem becomes much easier to solve. The next aim is to find tight upper and lower bounds to the problem by iteratively processing sequence of modified sub-problems. LR involves addressing two important issues; one is a strategic issue and the other a tactical issue. The strategic issue concerns the classification and relaxation of the constraints. The strategic question is of the form What constraints are to be relaxed? The tactical issue deals with the selection of a good technique for updating the Lagrange multipliers. The tactical questions are of the form, How the reduced problem can be solved? or How can we calculate an efficient bound?.

### 1.3.2 Relaxation of constraints

Before defining the general MIP problem, lets identify the following index sets:

$B = \{1, ..., |B|\}$ Index set for binary variables,

$I = \{|B| + 1, ..., |B| + |I|\}$ Index set for integer variables,

$C = \{|B| + |I| + 1, ..., |B| + |I| + |C|\}$ Index set for continues variables,

$N = B \cup I \cup C$ Index set for all variables.

Hence, the general MIP problem can be written as:

$P_0$ :

$min \sum_{j \in N} c_j x_j$

$s.t. \sum_{j \in N} a_{kj} x_j (\overset{\geq}{=}) d_k,$ k=1,...,m

$\sum_{j \in N} b_{lj} x_j (\overset{\geq}{=}) g_l,$ l=1,...,n

$x_j \in R^+$ iff $j \in C$

$x_j \in \{0, 1\}$ iff $j \in B$

$x_j \in Z^+$ iff $j \in I$

In the following of this subsection 1.3.2 ...

This initial problem $P_0$ is known as the master problem. Since this master problem is difficult to solve, we relax a set of constraints, $CO \in [1, m]$, by attaching Lagrange multipliers $\lambda_k \geq 0$. Then, this relaxed group of constraints are appended to the objective function and forms the following Lagrange Lower Bound Problem (LLBP):

$P_{L(\lambda)}$ :

$min \sum_{j \in N} x_j (c_j - \sum_{k=1}^{m} \lambda_k a_{kj}) + \sum_{k=1}^{m} \lambda_k d_k$

s.t. $\sum_{j \in N} b_{lj} x_j (\overset{\geq}{=}) g_l,$ l=1,...,n

$x_j \in R^+$ iff $j \in C$

$x_j \in \{0, 1\}$ iff $j \in B$

$x_j \in Z^+$ iff $j \in I$

The Lagrangian multipliers, $\lambda_k$, penalise the violation of the corresponding relaxed constraints introduced in the objective function. The selection of which set of contraints to be relaxed is a *strategic issue.*

After decomposing the master problem, we are interested in choosing the appropriate numerical values for the Lagrange multipliers (tactical issue) for the problem $P_{L(\lambda)}$. In particular, we are interested in finding the values of $\lambda$ that gives the maximum lower bound.[3] The Lagrange lower bound is also known as the Lagrange dual program.

$$max_{\lambda_k} \left\{ \begin{array}{l} min \sum\limits_{j \in N} x_j(c_j - \sum\limits_{k=1}^{m} \lambda_k a_{kj}) + \sum\limits_{k=1}^{m} \lambda_k d_k \\ \\ s.t. \sum\limits_{j \in N} b_{lj} x_j (\overset{\geq}{=}) g_l, \quad l = 1, ..., n \\ \\ x_j \in R^+ \quad iff \quad j \in C \\ \\ x_j \in \{0,1\} \quad iff \quad j \in B \\ \\ x_j \in Z^+ \quad iff \quad j \in I \end{array} \right\} \qquad (P_{Dual})$$

The best value for $\lambda_k$ is calculated by applying iterative updating techniques to the above system $(P_{dual})$. There are two well-known techniques that have been widely used: Subgradient Optimization and Multiplier Adjustment.

The estimation of good solution to NP-hard problems by using a non-exact method, like LR, does not depend only on the calculation of good lower bound. It is equally important to to calculate good solutions that are feasible and provede upper bounds to the master problem. We thus reduce the duality gap and provide tight bound for the optimal solution. The duality gap is defined as the relative difference between the lower bound and the upper bound. In ideal instances, the Lagrange lower bound is equal to the upper bound. The upper bounds are usually calculated by using a Lagrange heuristic (LH). An instant of a LH algorithm is to take the LLBP solution vector and to attempt to convert it to a feasible solution vector to the master problem.

### 1.3.3 Determination of the Lagrangian multipliers

There have been two main techniques that have been sucessfully applied for finding Lagrange multipliers in a wide variety of problem instances. There are the *subgradient optimization* and *multiplier adjustment*. Subgradient optimisation is an iterative procedure that, starting from an initial set of Lagrange Multipliers, attempts to improve the lower bound of the LLBP in a systematic way. Multiplier adjustment is also an iterative procedure, but modifies only one component of the multiplier in an iteration.

The literature suggests that *subgradient optimization* is the preferable method for general discrete optimisation problems. Subgradient is straight forward to implement and can be applied without modifications for different problem instances.

*Algorithmic Framework of Subgradient Optimisation*

Define $C_j$ as the cost coefficient vector of the LLBP $(P_{L(\lambda)})$. Hence,

$$C_j = c_j - \sum_{k=1} \lambda_k a_{kj}$$

[3]If the $P_0$ problem is max ..., then we are seeking for the minimum upper bound.

where j = 1,..,n (number of coefficients (variables)) and k = 1,...,m (number of constraints). The main steps that have to be followed to apply subgradient optimisation are set out below:

---

**STEP1:** *initialisation*

- Set $\pi$ which is a user-defined parameter, equal to 2. $(0 \leq \pi \leq 2)$
- Set the lower bounds to $-\infty$ and the upper bounds UB, $Z_{UB}$ to $+\infty$.
- Set N_LR = 0 number of Lagrange operations.
- Initialise the Lagrange multipliers $\lambda$.

**STEP2:** *calculate lower bound* **with subgradient method**

- Solve the LLBP($P_{L(\lambda)}$) for the current set of $\lambda_k$ to obtain the solution vector $X_j$ and the lower bound $Z_{LB}$. $(Z^t_{LB} = \{x_j\})$
- If the $Z_{LB} >$ LB, set LB = $Z_{LB}$.

**STEP3:** *calculate upper bound*

- Apply a Lagrange Heuristic to find a feasible upper bound $Z_{UB}$. If $Z_{UB} <$ UB, set UB = $Z_{LB}$.

**STEP4:** *Update the multipliers*

1. Calculate the Subgradients $G^t_k$ for current solution vector $X_j$.

$$G^t_k = d_k - \sum_{j \in N} a_{kj} x_j, \qquad k = 1, ...m$$

   If all $G_i \leq 0$ for each '$\geq$' constraint, then $Z_{LB}$ is feasible.

2. Define a scalar step size T.

$$T = \frac{\pi (Z_{UB} - Z_{LB})^2}{\sum\limits_{i=1}^{m} (G^t_k)}$$

3. Update the Lagrange Multipliers set

$$\lambda^{t+1}_k = max(0, \lambda^t_k + T G^t_k), \qquad k = 1, ..., m$$

**STEP5:** *Stopping criteria*

1. $\pi < 0.005$
2. (UB-LB) = 0.0

3. $\sum_{i=1}^{m}(G_k^t)^2 = 0$

If stopping rules are not stisfied then go to STEP 2.

---

The user-defined parameter, controls the step size T. In the case wherein the lower bound did not improve for 30 consecutive iterations, we half this parameter. Generally speaking, the smaller the value of this parameter, the smaller is the oscillation of the resulted lower bound (ZLB). In fact, when the value of the parameter is small, we are trying to improve the lower bound by searching on the "neighbourhood" of the LB.

There are three termination conditions of the algorithm. The algorithm terminates when the user-defined parameter becomes very small (i.e. 0.005), or when the dual gap (UB-LB) is equal to zero, or when the sum of squares of all the subgradients is equal to zero ($\sum_{i=1}^{m}(G_k^t)^2 = 0$). The last termination implies that all the constraints are perfectly satisfied and therefore all the Slack variables of the model are equal to zero.

## 2 Sidechain placing in Homology Modeling via Lagrangian Relaxation - ILP formulation from Kingsford et.al.2005

### 2.1 ILP formulation

If all pairwise energies between rotamers in positions i and j are non-positive, then we can remove all variables $x_{uv}$ with $u \in V_i$ and $E_{uv} = 0$, and modify the equality constraints

$$\sum_{u \in V_j} x_{uv} = x_{vv} \quad for\ j = 1, .., p \text{ and } \ v \in V/V_j$$

For each $V_j$ let $N^+(V_j)$ the set union of the $V_i$ for which there exists some $v \in V_i$ and $u \in V_j$ with $E_{uv} > 0$. Let $D'$ be the set of pairs {u,v} with $u \in V_j$ such that either $v \in N^+(V_j)$, or $v \notin N^+(V_j)$ but $E_{uv} < 0$. There will be edge variables $x_{uv}$ only for pairs in $D'$.

Our modified ILP is as follows:

$$Minimize\ \ E' = \sum_{u \in V} E_{uu} x_{uu} + \sum_{\{u,v\} \in D'} E_{uv} x_{uv}$$

subject to

$$\sum_{u \in V_j} x_{uu} = 1 \quad for\ j = 1, .., p$$

$$\sum_{u \in V_j} x_{uv} = x_{vv} \quad for\ j = 1, .., p \ \ and \ \ v \in N^+(V_j)$$

$$\sum_{u \in V_j : E_{uv} < 0} x_{uv} \le x_{vv} \quad for \;\; j = 1,..,p \;\; and \;\; v \notin N^+(V_j)$$

An inequality constraint is not included if the sum on the left-hand side is empty.

## 2.2 Lagrangian Relaxation of Kingsford-Formulation

$$max_{\lambda_{jv},\mu_{jv} \ge 0} \left\{ \begin{array}{l} min_{x_{uu},x_{uv}} \{ \sum\limits_{u \in V} E_{uu} x_{uu} \\ \\ + \sum\limits_{\{u,v\} \in D'} E_{uv} x_{uv} \\ \\ + \sum\limits_{j=1,...,p \; and \; v \in N^+(V_j)} \lambda_{jv} ( \sum\limits_{u \in V_j} x_{uv} - x_{vv}) \\ \\ + \sum\limits_{j=1,...,p \; and \; v \notin N^+(V_j)} \mu_{jv} ( \sum\limits_{u \in V_j : E_{uv} < 0} x_{uv} - x_{vv}) \} \\ \\ \\ \text{subject to:} \\ \\ \sum\limits_{u \in V_j} x_{uu} = 1 \;\; \text{for} \;\; j = 1,...,p \\ \\ x_{uu}, x_{uv} \; \in \; \{0,1\} \end{array} \right\} \qquad (P_{Dual})$$

### 2.2.1 Dimension of variables and constraints

State variables $x_{uu}$ and $x_{uv}$:

There are $|V| = n_1 + ... + n_p$ variables $x_{uu}$
and $|V|^2/2$ variables $x_{uv}$.

Lagrangian multipliers $\lambda_{jv}$ and $\mu_{jv}$:

There are $p * |V|$ multiplier variables $\lambda_{jv}$
and $p * |V|$ multiplier variables $\mu_{jv}$.

Constraints:

There are $p$ constraints
$\sum_{u \in V_j} x_{uu} = 1$ for $j = 1,...,p$.

For a normal protein with rotamers from a library like the Dunbrack-Library (see [7]) $p \approx 400$ and $n_i \approx 80$ for i=1..p. Thus we have $\approx 5 * 10^8$ state variables,
$\approx 1,2 * 10^7$ Lagrangian multipliers, and $\approx 400$ constraints

### 2.2.2 Implementation of the Lagrangian Relaxation

We can rewrite the following term in the energy function

$$\sum_{\{u,v\}\in D'} E_{uv}x_{uv}$$

as

$$\sum_{j=1,\dots,p:\ u\in V_j,\ v\in N^+(V_j)} E_{uv}x_{uv} + \sum_{j=1,\dots,p:\ v\notin N^+(V_j),\ u\in V_j : E_{uv}<0} E_{uv}x_{uv}$$

and

$\sum_{u\in V} E_{uu}x_{uu}$ as $\sum_{v\in V} E_{vv}x_{vv}$.

The algorithm consists of the following steps:

1. **initialisation of x, $\lambda$ and $\mu$.**
2. **for fix $\lambda$ and $\mu$ the energy function is minimized over $x_{uu}$ and $x_{uv}$.**
   That means each $x_{uu}$ and $x_{uv}$ are set (within the uncomplicating constraints) either to 0 or to 1 such that the relaxed energy function is minimized with fixed $\lambda$ and $\mu$.

$$z_{LR}(\lambda,\mu\geq 0) = min_{x_{vv},x_{uv}}\left\{\begin{array}{l}
\sum_{v\in V} E_{vv}x_{vv} - \sum_{j=1,\dots,p:\ v\in N^+(V_j)} \lambda_{jv}x_{vv} - \sum_{j=1,\dots,p:\ v\notin N^+(V_j)} \mu_{jv}x_{vv} \\
\\
+ \sum_{j=1,\dots,p:\ v\in N^+(V_j)} \sum_{u\in V_j} E_{uv}x_{uv} \\
\\
+ \sum_{j=1,\dots,p:\ v\notin N^+(V_j),\ u\in V_j:E_{uv}<0} E_{uv}x_{uv} \\
\\
+ \sum_{j=1,\dots,p:\ v\in N^+(V_j)} \sum_{u\in V_j} \lambda_{jv}x_{uv} \\
\\
+ \sum_{j=1,\dots,p:\ v\notin N^+(V_j)} \sum_{u\in V_j:E_{uv}<0} \mu_{jv}x_{uv} \\
\\
\text{subject to:} \\
\\
E_{v_{11}}x_{v_{11}} \ \dots\dots\dots\dots\dots\dots \lambda_{2v_{11}}x_{v_{11}} \text{ oder } \mu_{2v_{11}}x_{v_{11}} \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad . \\
\sum_{u\in V_j} x_{uu} = 1 \quad \text{for} \quad j=1,\dots,p \\
\\
x_{uu}, x_{uv} \ \in \ \{0,1\}
\end{array}\right\}$$

First $z_{LR}$ is optimized with respect to $x_{vv}$. There are only concerned the terms in the first line. We can divide the $v\in V$ into p classes $V_j, j=1...p$. Because we are minimizing we suppose $E_{vv}<0$.

There are $|V| = n_1 + ... + n_p$ elements in $\sum_{v \in V} E_{vv} x_{vv}$ and $(p-1) * |V|$ elements in $- \sum_{j=1,...,p:\ v \in N^+(V_j)} \lambda_{jv} x_{vv} - \sum_{j=1,...,p:\ v \notin N^+(V_j)} \mu_{jv} x_{vv}$

---

$j = 1:$

$E_{v_{11}v_{11}} x_{v_{11}v_{11}}$ ...................... $\lambda_{2v_{11}} x_{v_{11}v_{11}}$ oder $\mu_{2v_{11}} x_{v_{11}v_{11}}$

.

.

$\lambda_{pv_{11}} x_{v_{11}v_{11}}$ oder $\mu_{pv_{11}} x_{v_{11}v_{11}}$

.

.

$E_{v_{1n_1}v_{1n_1}} x_{v_{1n_1}v_{1n_1}}$ ................... $\lambda_{2v_{1n_1}} x_{v_{1n_1}v_{1n_1}}$ oder $\mu_{2v_{1n_1}} x_{v_{1n_1}v_{1n_1}}$

.

$\lambda_{pv_{1n_1}} x_{v_{1n_1}v_{1n_1}}$ oder $\mu_{pv_{1n_1}} x_{v_{1n_1}v_{1n_1}}$

---

.

.

---

$j = k:$

$E_{v_{k1}v_{k1}} x_{v_{k1}v_{k1}}$ ...................... $\lambda_{1v_{k1}} x_{v_{k1}v_{k1}}$ oder $\mu_{1v_{k1}} x_{v_{k1}v_{k1}}$

.

(not $\lambda_{kv_{k1}} x_{v_{k1}v_{k1}}$ oder $\mu_{kv_{k1}} x_{v_{k1}v_{k1}}$)

.

$\lambda_{pv_{k1}} x_{v_{k1}v_{k1}}$ oder $\mu_{pv_{k1}} x_{v_{k1}v_{k1}}$

.

.

$E_{v_{kn_k}} x_{v_{kn_k}v_{kn_k}}$ ................... $\lambda_{1v_{kn_k}} x_{v_{kn_k}v_{kn_k}}$ oder $\mu_{1v_{kn_k}} x_{v_{kn_k}v_{kn_k}}$

.

(not $\lambda_{kv_{kn_k}} x_{v_{kn_k}v_{kn_k}}$ oder $\mu_{kv_{kn_k}} x_{v_{kn_k}v_{kn_k}}$

.

$\lambda_{pv_{kn_k}} x_{v_{kn_k}v_{kn_k}}$ oder $\mu_{pv_{kn_k}} x_{v_{kn_k}v_{kn_k}}$

---

.

.

---

$j = p:$

$E_{v_{p1}v_{p1}} x_{v_{p1}v_{p1}}$ ...................... $\lambda_{1v_{p1}} x_{v_{p1}v_{p1}}$ oder $\mu_{1v_{p1}} x_{v_{p1}v_{p1}}$

.

$\lambda_{(p-1)v_{p1}} x_{v_{p1}v_{p1}}$ oder $\mu_{(p-1)v_{p1}} x_{v_{p1}v_{p1}}$

.

.

$E_{v_{pn_p}} x_{v_{pn_p}v_{pn_p}}$ ................... $\lambda_{1v_{pn_p}} x_{v_{pn_p}v_{pn_p}}$ oder $\mu_{1v_{pn_p}} x_{v_{pn_p}v_{pn_p}}$

.

$\lambda_{(p-1)v_{pn_p}} x_{v_{pn_p}v_{pn_p}}$ oder $\mu_{(p-1)v_{pn_p}} x_{v_{pn_p}v_{pn_p}}$

---

Determination of the $x_{vv}$:

In each class $V_j$ only one $x_{vv}$ is 1, the rest is 0.

```
/* j: residue, n[j]: number of rotamers in residue j */
$j, k, l = 0;$
$j = 1..p:$
    $for\ \ k = 1..n[j]:$
    $relaxedRotamerIntEnergy[j][k] = 0;$
        $relaxedRotamerIntEnergy[j][k] = relaxedRotamerIntEnergy[j][k] + E_{v_{jk}v_{jk}}$
        $for\ \ l = 1..p, l \neq j$
            $relaxedRotamerIntEnergy[j][k] = relaxedRotamerIntEnergy[j][k] - max(\lambda_{lv_{jk}}, \mu_{lv_{jk}})$

 /* now for all rotamers of all residues the relaxed internal energies are computed.
 Now for each residue j the rotamer with the minimal internal energy has to be chosen. */


$for\ \ j = 1..p:$
 /* Initialisation */
$RotWithMinRelaxIntEnergy[j] = 1;$
$MinRelaxIntEnergy[j] = relaxedRotamerIntEnergy[j][1];$
    $for\ \ k = 1..n[j]:$
        $if\ relaxedRotamerIntEnergy[j][k]\ <\ MinRelaxIntEnergy[j]$
            $\mathbf{RotWithMinRelaxIntEnergy[j] = k;}$
            $MinRelaxIntEnergy[j] = relaxedRotamerIntEnergy[j][k];$
        $endif$

 /* now for all variables $x_{vv}$ are set. */
$for\ \ j = 1..p:$
    $for\ \ k = 1..n[j]:$
 /* Initialisation */
    $x_{vv}[j][k] = 0;$

$for\ \ j = 1..p:$
    $for\ \ k = 1..n[j]:$
        $if\ RotWithMinRelaxIntEnergy[j] == k$
            $x_{vv}[j][k] = 1;$
        $endif$
```

Determination of the $x_{uv}$:

$$x_{uv} = \begin{cases} 1, & \text{if } x_{uu} = 1 \text{ and } x_{vv} = 1 \text{ (from determination above)} \\ 0, & \text{otherwise} \end{cases}$$

3. **The Lagrangian multipliers are updated**

To update the Lagrangian multipliers $\lambda$ and $\mu$ we have to identify the variables in STEP4 of the subgradient algorithm in 1.3.3.

**STEP4:** *Update the multipliers*

(a) Calculate the Subgradients $G^t_{k'}$ for current solution vector $X_{j'}$.

$$G^t_{k'} = d_{k'} - \sum_{j' \in N} a_{k'j'} x_{j'}, \qquad k' = 1, ...m$$

(b) Define a scalar step size T.

$$T = \frac{\pi(Z_{UB} - Z_{LB})}{\sum} \overset{m}{\underset{i=1}{}} (G^t_{k'})^2$$

(c) Update the Lagrange Multipliers set

$$\lambda^{t+1}_{k'} = max(0, \lambda^t_{k'} + TG^t_{k'}), \qquad k' = 1, ..., m$$

The variable sets $d_{k'}$ and $a_{k'j'}$ in (a) are from the complicating constraints (see 1.3.2). We can write our complicated constraints

$$\sum_{u \in V_j} x_{uv} = x_{vv} \quad for\ j = 1,..,p \ \ and \ \ v \in N^+(V_j)$$

$$\sum_{u \in V_j : E_{uv} < 0} x_{uv} \leq x_{vv} \quad for \ \ j = 1,..,p \ \ and \ \ v \ \notin \ N^+(V_j)$$

as:

```
$for \ \ j = 1..p :$
  $for \ \ k = 1..n[j] :$
    $for \ \ l = 1..p, l \neq j :$
     $if \ \ v \in N^+(V_j)$
       $\sum_{r=1}^{n[l]} x_{uv}[l][r][j][k] - x_{vv}[j][k] = 0$
     $elseif \ v \notin \ N^+(V_j)$ and $u \in V_j : E_{uv} < 0$
       $\sum_{r=1}^{n[l]} x_{uv}[l][r][j][k] - x_{vv}[j][k] \leq 0$
```

<u>variable substitution:</u>

elements of V:

$j = 1..p$

$k = 1..n[j]$

for each element of V there is now an edge defined to all other elements of V except to those which are in the same class $V_j$ :

$l = 1..p, l \neq j$

$r = 1..n[l]$

$x_{uv}[l][r][j][k] \rightarrow x[l][r][j][k]$

$x_{vv}[j][k] \qquad \rightarrow x[p+1][0][j][k]$

Some explanations concerning the variables:
The following variables are defined through the following indices:

$x_{uv} : [l][r][j][k], (j = 1..p; k = 1..n[j]; l = 1..p, l \neq j; r = 1..n[l])$
$x_{vv} : [j][k], (j = 1..p; k = 1..n[j])$
$m$ complicating constraints: $[j][k][l], (j = 1..p; k = 1..n[j]; l = 1..p, l \neq j)$
$a_{j'k'} : [l_1][r_1][j_1][k_1]$(for j': index of state variables) and
$[j_2][k_2][l_2]$(for: k': index for the m complicating constraints)

Computation of $a_{j'k'}$:

```
$for\ \ j_1 = 1..p :$
  $for\ \ k_1 = 1..n[j_1] :$
    $for\ \ l_1 = 1..p, l_1 \neq j_1 :$
     $if\ \ v \in N^+(V_j)$
       $for\ \ r_1 = 1..n[l_1] :$
         $for\ \ j_2 = 1..p :$
          $for\ \ k_2 = 1..n[j_2] :$
            $for\ \ l_2 = 1..p, l_2 \neq j_2 :$
              $a[l_1][r_1][j_1][k_1][l_2][j_2][k_2] = 1$ /* factors of $x_{uv}$*/
              $a[p+1][0][j_1][k_1][l_2][j_2][k_2] = -1$ /* factors of $x_{vv}$*/
     $elseif\ \ v \notin N^+(V_j)$ and $u \in V_j : E_{uv} < 0$
       $for\ \ r_1 = 1..n[l_1] :$
         $for\ \ j_2 = 1..p :$
          $for\ \ k_2 = 1..n[j_2] :$
            $for\ \ l_2 = 1..p, l_2 \neq j_2 :$
              $a[l_1][r_1][j_1][k_1][l_2][j_2][k_2] = 1$ /* factors of $x_{uv}$*/
              $a[p+1][0][j_1][k_1][l_2][j_2][k_2] = -1$ /* factors of $x_{vv}$*/
     $else$
       $for\ \ r_1 = 1..n[l_1] :$
         $for\ \ j_2 = 1..p :$
          $for\ \ k_2 = 1..n[j_2] :$
            $for\ \ l_2 = 1..p, l_2 \neq j_2 :$
              $a[l_1][r_1][j_1][k_1][l_2][j_2][k_2] = 0$
              $a[p+1][0][j_1][k_1][l_2][j_2][k_2] = 0$
```

We identify $d_{k'} = 0$.
We have to compute the vector $G^t_{k'}$, k'=1,...,m. m is the number of complicating constraints, which are integrated into the energy function. Thus m is also the number of Lagrangian multipliers.

Computation of the vector $G^t_{k'}$, k'=1,...,m :

Calculate the Subgradients $G^t_{k'}$ for current solution vector $X_{j'}$.

$$G^t_{k'} = d_{k'} - \sum_{j' \in N} a_{k'j'} x_{j'}, \qquad k' = 1, ...m$$

Because $d_{k'} = 0$:

$$G^t_{k'} = - \sum_{j' \in N} a_{k'j'} x_{j'}, \qquad k' = 1, ...m$$

Because k' = 1,...m is equivalent to

```
$for\ \ j_2 = 1..p:$
  $for\ \ k_2 = 1..n[j_2]:$
    $for\ \ l_2 = 1..p, l_2 \neq j_2$
```

we write $G_t[l_2][j_2][k_2]$ instead of $G^t_{k'}$.

```
/* initialization */
        $for\ \ j_2 = 1..p:$
          $for\ \ k_2 = 1..n[j_2]:$
            $for\ \ l_2 = 1..p, l_2 \neq j_2:$
              $G_t[l_2][j_2][k_2] = 0$

/* $- \sum_{j' \in N} a_{k'j'} x_{uv}$ */
$for\ \ j_1 = 1..p:$
  $for\ \ k_1 = 1..n[j_1]:$
    $for\ \ l_1 = 1..p, l_1 \neq j_1:$
     $if\ \ v \in N^+(V_j)$
       $for\ \ r_1 = 1..n[l_1]:$
         $for\ \ j_2 = 1..p:$
           $for\ \ k_2 = 1..n[j_2]:$
             $for\ \ l_2 = 1..p, l_2 \neq j_2:$
               $G_t[l_2][j_2][k_2] = G_t[l_2][j_2][k_2] - a[l_1][r_1][j_1][k_1][l_2][j_2][k_2] * x[l_1][r_1][j_1][k_1]$
     $elseif\ \ v \notin N^+(V_j)$ and $u \in V_j : E_{uv} < 0$
       $for\ \ r_1 = 1..n[l_1]:$
         $for\ \ j_2 = 1..p:$
           $for\ \ k_2 = 1..n[j_2]:$
             $for\ \ l_2 = 1..p, l_2 \neq j_2:$
               $G_t[l_2][j_2][k_2] = G_t[l_2][j_2][k_2] - a[l_1][r_1][j_1][k_1][l_2][j_2][k_2] * x[l_1][r_1][j_1][k_1]$
/* $- \sum_{j' \in N} a_{k'j'} x_{vv}$ */
$for\ \ j_1 = 1..p:$
  $for\ \ k_1 = 1..n[j_1]:$
         $for\ \ j_2 = 1..p:$
           $for\ \ k_2 = 1..n[j_2]:$
             $for\ \ l_2 = 1..p, l_2 \neq j_2:$
               $G_t[l_2][j_2][k_2] = G_t[l_2][j_2][k_2] - a[p+1][0][j_1][k_1][l_2][j_2][k_2] * x[p+1][0][j_1][k_1]$
```

4. **go to 1.**

## References

[1] Lamport, L., *LaTeX : A Documentation Preparation System User's Guide and Reference Manual*, Addison-Wesley Pub Co., 2nd edition, August 1994.

[2] Nemhauser and Wolsey, 1988

[3] Althaus, Kohlbacher et.al.2002

[4] Eriksson, O., Zhou, Y. and Elofsson, A. (2001) Side-chain positioning as an integer programming problem. In *Proceedings of 1st Workshop on Algorithms in BioInformatics*, BRICS, University of Aarhus, Denmark, pp.129-141

[5] Caprara and Lancia 2002

[6] C Siamitros, G Mitra and C Poojari *Revisiting Lagrange Relaxation for Processing Large-Scale Mixed Integer Programming Problems*, CTR/27/04 April 2004, Department of Mathematical Sciences and Department of Economics and Finance, Brunel University, Uxbridge, Middlesex, UB8 3PH, www.carisma.brunel.ac.uk

[7] Dunbrack, Rotamer library...

[8] Kingsford, C.L., Chazelle, B. and Singh, Mona (2005) Solving and analysing side-chain positioning problems using linear and integer programming. *Bioinformatics*, **21 No.7**, 1028-1036

[9] Lasters, I., De Mayer, M. & Desmet, J.: Enhanced dead-end elimination in the search for the global minimum energy conformation of a collection of protein side chains. Protein Eng. **8** (1995) 815-822