# Protein similarity search under mRNA structural constraints: application to targeted selenocysteine insertion

Rolf Backofen* N.S. Narayanaswamy† Firas Swidan
Oettingenstrasse 66, 80538 Munich, Germany
Tel: +49 89 2178 2213 Fax: +49 89 2178 2238
Email: {backofen,swamy,swidan}@informatik.uni-muenchen.de

**Abstract**

Selenocysteine is the 21th amino acid, which occurs in all kingdoms of life. Selenocysteine is encoded by the STOP-codon UGA. For its insertion, it requires a specific mRNA sequence downstream the UGA-codon that forms a hairpin like structure (called Sec insertion sequence (SECIS)). We consider the computational problem of generating new amino acid sequences containing selenocysteine. This requires to find an mRNA sequence that is similar to the SECIS-consensus, is able to form the secondary structure required for selenocysteine insertion, and whose translation is maximally similar to the original amino acid sequence. We show that the problem can be solved in linear time when considering the hairpin-like SECIS-structure (and, more generally, when considering a structure that does not contain pseudoknots).
**Keywords: Selenocysteine, SECIS, Protein Engineering**

# 1 Introduction

## 1.1 Selenoprotein and Selenocysteine Insertion

*Selenocysteine* (Sec) is a rare amino acid, which was discovered as the 21st amino acid [Böck et al., 1991] about a decade ago. Proteins containing selenocysteine are consequently called *selenoproteins*. The discovery of selenocysteine was another clue to the complexity and flexibility of the mRNA translation mechanism. Selenocysteine is encoded by the UGA-codon, which is usually a STOP-codon encoding the end of translation by the ribosome. It has been shown [Böck et al., 1991] that in the case of selenocysteine, termination of translation is inhibited in the presence of a specific mRNA sequence in the 3'-region after the UGA-codon that forms a hairpin like structure (called Sec insertion sequence (SECIS), see Figure 1). Selenoproteins occur in all domains of life.

Selenium is very important for human health. The more than 30 known human selenoproteins are essential components for several major metabolic pathways, including antioxidant defense systems and immune function (for an review, see [Brown and Arthur, 2001]). Albeit the process of selenocysteine-insertion in mammalian selenoproteins is not sufficiently understood yet, it is known that it requires the existence of a SECIS-element in the 3' UTR region of the mRNA with a distance from the UGA-element that naturally varies from 500 to 5300 nucleotides [Low and Berry, 1996]. It has been shown that selenocysteine insertion requires some additional factor, namely the SECIS-binding-protein (SBP2) [Copeland et al., 2000], albeit the exact interaction between SBP2 and the SECIS-element is not yet completely understood. In addition, a special elongation factor mSelB has been proposed recently [Fagegaltier et al., 2000]. Corresponding the specifity of the 3' UTR-region, it has been shown that the 3'-UTR of three different selenoproteins are interchangeable, albeit there is only little sequence homology between their SECIS-elements. But the corresponding predicted structures show a high similarity (see [Low and Berry, 1996] for an review). In archaea,

*Partially supported by the DFG within the national program SPP 1087 "Selenoprotein – Biochemische Grundlagen und klinische Bedeutung"
†Supported by DFG Grant No. Jo 291/2-1

the SECIS-elements are located in the 3' UTR region of mRNA like in the case of eukaryotes (for an review see [Rother et al., 2001]).

In bacteria, the situation is quite different. The SECIS-element is located immediately downstream the UGA codon [Zinoni et al., 1990]. A displacement of the SECIS-element by more than one codon, or a displacement not preserving the reading-frame results in a drastic reduction of selenocysteine insertion efficiency [Heider et al., 1992]. For *E.coli*, the mechanism of selenocysteine insertion is well understood, and all corresponding factors are identified [Sawers et al., 1991]. The selenoprotein synthesis requires the products of four genes *selA–selD*, which encode a selenocysteine synthase (SELA), a special elongation factor SELB, a specific tRNA for selenocysteine ($tRNA^{Sec}$, which is the product of gene *selC*), and a selenophospate synthetase (SELD), required to produce selenophospate (the selenium donor to the tRNA) from selenide (see [Böck et al., 1991] for an review). SELB [Forchhammer et al., 1989] is a protein consisting of 4 domains, where the three N-terminal domains are homologous to the elongation factor Tu (EF-Tu). The 4th C-terminal domain shows no known homology and is the binding site for the SECIS-element [Kromayer et al., 1996]. So far, the structure of SELB is unsolved (only a structural model for the N-terminal domains homologous to EF-Tu exists [Hilgenfeld et al., 1996]).

Since the selenoprotein biosynthesis is a complex organism specific pathway, there are some barriers for the heterologous expression of genes encoding selenoprotein. On the positive side, it was found that there is a high degree of conservation of the selenoprotein biosynthesis throughout the enterobacteria [Heider et al., 1991].

There has been quite some effort to characterize the nature of the SECIS-element for *E.coli*. The lower region of the hairpin, which comprises the first 11 nucleotides following the UGA codon is not required for SelB binding *in vitro* [Kromayer et al., 1996] and has been suggested to function by enhancing the stability of the loop structure and/or to prevent binding of release factor 2 [Hüttenhofer et al., 1996]. It has been shown that deviations in the primary sequence of this distal part of the SECIS did not disturb UGA readthrough [Heider et al., 1992, Liu et al., 1998], but variations in its length (deletion or extension of three or six nucleotides) significantly reduced or abolished UGA decoding [Heider et al., 1992, Liu et al., 1998]. The upper part consisting of 17 nt of *fdhF* mRNA (18 nt of *fdnG* mRNA) is much more specific in primary sequence and secondary structure. This region is recognized and bound by SELB [Baron et al., 1993, Kromayer et al., 1996]. In addition, binding specifity of SELB (or more specificly of the 4th domain of SELB) has been investigated by in vitro selection approaches (artificial evolution). These experiments indicate that there is some variation of mRNA sequences that are bound [Klug et al., 1997, Klug et al., 1999]. Furthermore, there have been successful efforts of generating mutants of SelB that where capable of recognizing mutated SECIS-elements that are not recognized by the original SelB [Kromayer et al., 1999].

Selenocysteine is more reactive than cysteine. There has been quite some interest in comparing the enzymatic activity of proteins containing selenocysteine, and similar proteins not containing selenocysteine. On the one hand, it is simple to replace Sec by Cys. If the replaced selenocysteine is in the active site, this usually results in a reduction of enzymatic properties. Another source for comparison are homologous versions in different organisms (for an review, see [Stadtman, 1996]). For the other direction, namely the production of artificial selenoproteins by genetic methods, there were only two successful experiments. In both case, cysteine was replaced by selenocysteine by inserting a SECIS-element at the proper position. The inserted SECIS-element was identical to the wild type in [Heider and Böck, 1992], and engineered in [Hazebrouck et al., 2000].

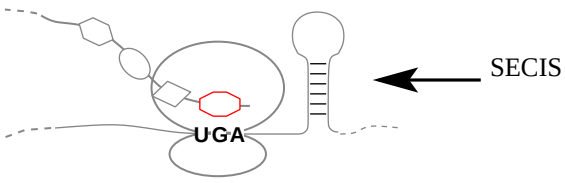



Figure 1: Translation of mRNA requires a SECIS-element in case of selenocysteine.

Concerning bioinformatics approaches in the area of selenoproteins, we have the following situation. An

obvious problem is to search for existing but unknown selenoproteins. Since both sequence and structure is important for selenocysteine incorporation, and the different selenoproteins do not have a pronounced consensus sequence (on the amino acid level), the usually BLAST and FASTA search tools will not have enough selectivity. Therefore, new algorithms have to be devised. An example is SECISearch [Kryukov et al., 1999], which searches for mammalian selenoproteins by recognizing the corresponding SECIS-element using a hierarchical approach. First, one searches for sequences that satisfy the SECIS-consensus sequence, and allow for the secondary structure required for the SECIS-element. In an additional step, the minimal free energy of the sequences is estimated using the RNAfold program of the Vienna-RNA-Package [Hofacker et al., 1994]. The program is capable of finding all the known mammalian SECIS elements. On the other hand, the authors also noticed that a significant portion of SECIS-elements found by the program are pseudo-SECISes.

However, we consider a different problem, namely modifying existing bacterial proteins (or proteins that can be expressed in a bacteria, usually *E.coli*) such that selenocysteine is incorporated instead of another amino acid (usually cysteine) at a given position. To achieve this, the mRNA of the protein has to be modified such that a SECIS is formed at the required position. In most cases, the produced SECIS element cannot be a wild-type SECIS, but must be an engineered SECIS element.

There are two possible reasons for wanting such modifications. First, it provides a possibility of generating new, functionally modified selenoproteins. Experimentally, this has been done successfully for *E. coli* in [Hazebrouck et al., 2000], where the mRNA sequence of a protein not containing selenocysteine was engineered (by hand) such that it forms a *E. coli* SECIS (while preserving maximum similarity on the amino acid level). The engineered protein now had a selenocysteine in place of a catalytic cysteine (at position 41), and showed a 4-fold enhanced catalytic activity. Second, and more important, seleno-variants of proteins could be used for the phase determination in an X-ray crystallography (or in the NMR-analysis) of a given protein.

In this paper, we consider the computational properties of substituting one position by a selenocysteine in a given amino acid sequence. Let $S = S_1 \dots S_{3n}$ be the consensus of the SECIS-element, and let $A = A_1 \dots A_n$ be the original amino acid sequence following the position where we wish to insert selenocysteine. By modifying an mRNA sequence to insert a selenocysteine the sequence $A$ may be modified. We have to find an appropriate mRNA sequence $N = N_1 \dots\dots N_{3n}$, which is a SECIS element and encodes an amino acid sequence $A' = A'_1 \dots A'_n$ that has maximum similarity with $A$. Thus, we have the following picture:

$$\begin{array}{lccccc}
A = & \underset{\wr}{A_1} & \dots & \underset{\wr}{A_i} & \dots & \underset{\wr}{A_n} \\
A' = & \underset{\uparrow}{A'_1} & \dots & \underset{\uparrow}{A'_i} & \dots & \underset{\uparrow}{A'_n} \\
N = & \overbrace{\underset{\wr}{N_1}\underset{\wr}{N_2}\underset{\wr}{N_3}} & \dots & \overbrace{\underset{\wr}{N_{3i-2}}\underset{\wr}{N_{3i-1}}\underset{\wr}{N_{3i}}} & \dots & \overbrace{\underset{\wr}{N_{3n-2}}\underset{\wr}{N_{3n-1}}\underset{\wr}{N_{3n}}} \\
S = & S_1\, S_2\, S_3 & \dots & S_{3i-2}S_{3i-1}S_{3i} & \dots & S_{3n-2}S_{3n-1}S_{3n}.
\end{array}$$

The similarities on both the amino acid ($A_i \sim A'_i$) and nucleotide level ($N_j \sim S_j$) will be measured by functions $F_{A_i}^{S_{3i-2}S_{3i-1}S_{3i}}(N_{3i-2}N_{3i-1}N_{3i})$, and we search an $N$ that maximizes

$$\sum_{i=1}^{n} F_{A_i}^{S_{3i-2}S_{3i-1}S_{3i}}(N_{3i-2}N_{3i-1}N_{3i}),$$

(referred to as *similarity*) and satisfies the constraints given by the SECIS structure. These constraints are given by the bonds formed by the structure. If there is a bond between position $r$ and $s$ in the SECIS, then every valid $N$ has to satisfy that $N_r$ and $N_s$ are complementary.

Using the fact that the structure of the SECIS element is a hairpin, we show that the described problem can be solved in linear time. The property that makes the problem computationally easy is that in the linear embedding of the hairpin structure, no edges cross. We have shown that it is $NP$-hard to solve the problem optimally if every linear embedding has crossing edges [Backofen et al., ]. This case can occur if we have pseudoknots as an underlying structure. Pseudoknots as an mRNA-structure are not important for selenocysteine insertion, but e.g. for programmed frameshifts, which allow to encode two different

amino acid sequences in one mRNA sequence. This mechanism has been identified in viruses as well as in prokaryotes and eukaryotes [Farabaugh, 1996, Giedroc et al., 2000].

# 2 Preliminaries

We follow graph theoretic notations and definitions as presented in the standard text by Harary [Harary, 1972]. A labeled graph $G(V, E, \mathrm{Lab})$ is an undirected graph $G(V, E)$ along with a function $\mathrm{Lab} : E \to X$, where $X$ is a finite set whose elements are called *edge labels*. In this paper, $X = \{1, -1\}$. An undirected graph is said to be *outer-planar* if it can be drawn satisfying the following conditions: 1.) all the vertices lie on a line, 2) all the edges lie on one side of the line, and 3.) two edges that intersect in the drawing do so only at their end points. Such a drawing is called an outer-planar embedding of the graph.
The solution to the problem we address in this paper is a string whose letters are from the set $\{A, C, G, U\}$. The elements of this set are referred to as *nucleotides*, and an element of $\{A, C, G, U\}^3$ is referred to as a *codon*. $A$ and $U$ are complements of each other and so are $G$ and $C$ (forming the standard Watson-Crick bonds). In addition, there are the non-standard bonds formed by $G$ and $U$, which implies that we consider $G$ and $U$ as complements of each other, too. The complement set of a variable is denoted by the superscript $C$, i.e., $A^C = \{U\}, C^C = \{G\}, G^C = \{U, C\}, U^C = \{G, A\}$. Now our problem can be stated as follows:

**Input:** An edge labeled graph $G = (V, E, \mathrm{Lab})$ on $3n$ vertices, $V = \{v_1, \ldots, v_{3n}\}$. For every edge $\{v_k, v_l\} \in E$, the label $\mathrm{Lab}(v_k, v_l)$ is either 1 or $-1$. If $\mathrm{Lab}(v_k, v_l) = +1$, then we have a bond between the nucleotides at positions $k$ and $l$ in the mRNA. Otherwise, there must not be a bond between positions $k$ and $l$. $n$ functions, $f_1, \ldots, f_n$ are also part of the input. $f_i$ is associated with $\{v_{3i-2}, v_{3i-1}, v_{3i}\}, 1 \leq i \leq n$, and is a function defined on $\{A, C, G, U\}^3$ and takes rational values.

**Output:** Find vector $(N_1, \ldots, N_{3n}) \in \{A, C, G, U\}^{3n}$ s.t. $N_k$ is assigned to $v_k$ and

1. $\{v_k, v_l\} \in E(G)$ and $\mathrm{Lab}(v_k, v_l) = 1$ implies that $N_k \in N_j^C$.
2. $\{v_k, v_l\} \in E(G)$ and $\mathrm{Lab}(v_k, v_l) = -1$ implies that $N_k \notin N_j^C$.
3. $\sum_{i=1}^{n} f_i(N_{3i-2}, N_{3i-1}, N_{3i})$ is maximized. This function is referred to as the *similarity* of the assignment at which it is evaluated.

We denote the above problem by $MRSO(G, f_1, \ldots, f_n)$. The short form MRSO stands for *mRNA Structure Optimization*. In the following, we will refer to conditions 1 and 2 together as the *complementarity condition*. We will call edges that are labeled with 1 *bonds*, and edges labeled by $-1$ *prohibited bonds*. Prohibited bonds are necessary since the SECIS also needs some bulged nucleotides. Since it is not necessary to add prohibited bonds for nodes that are part of a bond, we assume in the following that $\{v_k, v_l\} \in E(G)$ and $\mathrm{Lab}(v_k, v_l) = -1$ implies that $v_k, v_l$ are not part of a bond (i.e., there is no node $v$ different from $v_k, v_l$ such that $(v_k, v) \in E(G) \wedge \mathrm{Lab}(v_k, v) = 1$ or $(v_l, v) \in E(G) \wedge \mathrm{Lab}(v_l, v) = 1$).

**About Graphs:** $G$ is referred to as the *structure graph*. A structure graph that specifies only bonds is called an *RNA-graph*. For RNA-graphs, the degree of every node is at most one. Given a structure graph $G$, we define the *implied graph* $G^{\mathrm{impl}}$ to be a graph on the vertices $\mathrm{V}(G^{\mathrm{impl}}) = \{u_1, \ldots, u_n\}$ with

$$\mathrm{E}(G^{\mathrm{impl}}) = \left\{ \{u_i, u_j\} \;\middle|\; \begin{array}{l} \exists r \in \{3i-2, 3i-1, 3i\} : \\ \quad \exists s \in \{3j-2, 3j-1, 3j\} : (v_r, v_s) \in E(G) \end{array} \right\}.$$

Note that in the implied graph, every node has degree at most 3 if the input graph $G$ is a graph with degree at most 1.[1] This means that up to 3 edges can be emerging from a node in the implied graph. We follow the convention that, independent of the subscript or superscript, $u$ denotes a vertex from $\mathrm{V}(G^{\mathrm{impl}})$ and $v$ denotes a vertex from $G$. Given $I \subseteq \{1..n\}$, $U_I$ denotes $\{u_j \mid j \in I\}$. $\mathrm{E}(G^{\mathrm{impl}})|_I$ denotes the edge set of the induced subgraph of $G^{\mathrm{impl}}$ on $U_I$. $\mathrm{E}(G)||_I$ denotes the edge set of the induced subgraph of $G$ on $\{v_{3i-2}, v_{3i-1}, v_{3i} \mid i \in I\}$.

[1] i.e., an RNA-graph, or a graph with at most one prohibited bond for each node.

**About Codons:** Given a sequence of codons $L_1 \ldots L_n$, let $N_1 \ldots N_{3n}$ be the corresponding nucleotide sequence obtained by replacing $L_i$ by the corresponding nucleotide representation $N_{3i-2}N_{3i-1}N_{3i}$. $L_1 \ldots L_n$ is said to *satisfy* $E(G)$ iff the corresponding nucleotide representation $\{N_{3i-2}N_{3i-1}N_{3i}\}_{i=1}^{n}$ satisfies the complementarity conditions for $G$. $L_i$ and $L_j$ are said to be *valid for* $\{u_i, u_j\}$ *w.r.t.* $E(G)$ if the corresponding nucleotides $N_{3i-2}N_{3i-1}N_{3i}$ and $N_{3j-2}N_{3j-1}N_{3j}$ satisfies the complementarity condition imposed by $E(G)$.

# 3 Algorithm for SECIS-like Structures

We present a linear time recursive algorithm to solve MRSO when $G^{\text{impl}}$ is outer-planar *and* every node in $G$ has degree at most 1.[2] The hairpin shape of the SECIS is captured by the outer-planarity of $G^{\text{impl}}$. The algorithm is based on a recurrence relation that we prove in this section. Also, the algorithm solves MRSO even when the functions can take arbitrary values.

We fix $u_1, \ldots, u_n$ as the ordering of the vertices from left to right on the line in an outer-planar embedding of $G^{\text{impl}}$. For each $1 \leq i \leq n$, let $f_i$ be the function associated with $u_i$. Having fixed an embedding of the graph on the line, we do not distinguish between a vertex and its index. That is, the interval $[i, \ldots, i+k]$ denotes the set of vertices $\{u_i, \ldots, u_{i+k}\}$.

We now define the notion of compatibility between codons $L_i$ and $L_j$ assigned to vertices $u_i$ and $u_j$, respectively. We denote compatibility by $\equiv_{E(G)}$.[3] The three lines in $\equiv$ serve to indicate that the complementarity conditions dictated by $E(G)$ should be satisfied. For $1 \leq i, j \leq n$, define $(i, L_i) \equiv_{E(G)} (j, L_j)$ by

$$(i, L_i) \equiv_{E(G)} (j, L_j) = \begin{cases} true & \text{if } \{u_i, u_j\} \notin \mathrm{E}(G^{\text{impl}}) \\ true & \text{if } \{u_i, u_j\} \in \mathrm{E}(G^{\text{impl}}) \text{ and} \\ & L_i \text{ and } L_j \text{ are } \textit{valid} \text{ for } \{u_i, u_j\} \text{ w.r.t. } E(G) \\ false & otherwise, \end{cases}$$

$(i, L_i) \not\equiv_{E(G)} (j, L_j)$ denotes that $(i, L_i) \equiv_{E(G)} (j, L_j)$ is false. We now define the following function:

$$w(i, i+k, L_i, L_{i+k}) = \max_{L_{i+1}, \ldots, L_{i+k-1}} \left\{ \sum_{i \leq j \leq i+k} f_j(L_j) \mid \begin{array}{l} L_i, \ldots, L_{i+k} \\ \text{satisfies } \mathrm{E}(G)|_{[i..i+k]} \end{array} \right\}.$$

If the set over which the maximum is taken is empty, the value of the function is considered as $-\infty$. This function forms the central part of our algorithm to solve the problem because $\max_{L_1, L_n} w(1, n, L_1, L_n)$ is the value that we are interested in computing. The base cases for $w$ is the following:

$$w(i, i+1, L_i, L_{i+1}) = \begin{cases} f_i(L_i) + f_{i+1}(L_{i+1}) & \text{if } (i, L_i) \equiv_{E(G)} (i+1, L_{i+1}) \\ -\infty & \text{otherwise.} \end{cases}$$

We solve the problem on an interval $[i..i+k]$ by splitting $[i..i+k]$ into two parts and solving the resulting subproblems. If there is no edge between $i$ and any vertex in the interval $[i..i+k-1]$, the interval is split into $[i..i+1]$ and $[i+1..i+k]$. Otherwise, we choose the farthest vertex $p$ in $[i..i+k-1]$ which is adjacent to $i$. The edge $(i, p)$ is called the *maximal* edge in $\mathrm{E}(G^{\text{impl}})|_{[i..i+k-1]}$. Then, the interval is split into $[i..p]$ and $[p..i+k]$ (see Figure 2). Hence, define $\mathrm{next}(i, i+k)$ for $k \geq 2$ by

$$\mathrm{next}(i, i+k) = \begin{cases} i+r & \text{if } \{i, i+r\} \text{ is maximal in } \mathrm{E}(G^{\text{impl}})|_{[i..i+k-1]} \\ i+1 & \text{otherwise.} \end{cases}$$

[2] This implies that we may add only one prohibited bond for every node in $G$, which is sufficient for SECIS structures.

[3] **Note:** To read, associate the symbol $\equiv$ with the word *compatible*.

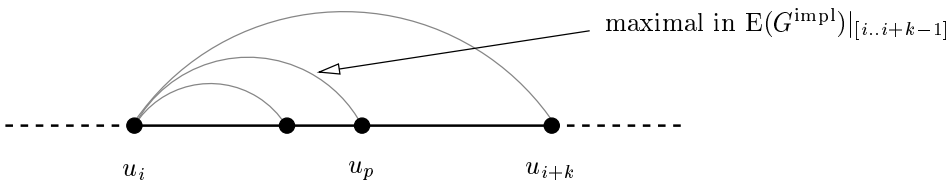



Figure 2: Def. of $\mathrm{next}(i, i+k)$

**Theorem 3.1 (Recurrence)** *Let $(G, f_1, \ldots, f_n)$ be instance of the MRSO. For $k \geq 2, 1 \leq i \leq n-k$, let $p = \mathrm{next}(i, i+k)$. Then*

$$w(i,i+k,L_i,L_{i+k}) = \begin{cases} -\infty & (i,L_i) \not\equiv_{E(G)} (i+k,L_{i+k}) \\ \max\limits_{L_p} \begin{pmatrix} & w(i,p,L_i,L_p) \\ + & w(p,i+k,L_p,L_{i+k}) \\ - & f_p(L_p) \end{pmatrix} & \textit{if } (i,L_i) \equiv_{E(G)} (i+k,L_{i+k}), \end{cases}$$

**Proof.** For the case when $(i, L_i) \not\equiv_{E(G)} (i+k, L_{i+k})$, the result is true because we follow the convention that the maximum over an empty set is $-\infty$. So we consider the case when $(i, L_i) \equiv_{E(G)} (i+k, L_{i+k})$. Let $k \geq 2$. We wish to evaluate,

$$\max_{L_p} \big( \; w(i,p,L_i,L_p) + w(p,i+k,L_p,L_{i+k}) - f_p(L_p). \; \big)$$

The above term is equal to

$$= \max_{L_p} \left\{ \begin{array}{l} \max\limits_{L_{i+1},\ldots,L_{p-1}} \big( \sum\limits_{i \leq j \leq p} f_j(L_j) \mid L_i, \ldots, L_p \text{ satisfies } \mathrm{E}(G)||_{[i..p]} \big) \\ + \max\limits_{L_{p+1},\ldots,L_{i+k-1}} \big( \sum\limits_{p \leq j \leq i+k} f_j(L_j) \mid L_p, \ldots, L_{i+k} \text{ satisfies } \mathrm{E}(G)||_{[p..i+k]} \big) \\ - f_p(L_p) \end{array} \right\}$$

Now we show that this equals $\max\limits_{L_{i+1},\ldots,L_{i+k-1}} \left\{ \sum\limits_{i \leq j \leq i+k} f_j(L_j) \mid \begin{array}{l} L_i, \ldots, L_{i+k} \\ \text{satisfies } \mathrm{E}(G)||_{[i..i+k]} \end{array} \right\}$

We first observe that there are no edges between vertices in the interval $[i+1..p-1]$ to vertices in the interval $[p+1, i+k]$. This is because we have assumed as input an outer-planar drawing of $G^{\mathrm{impl}}$. Therefore, the above two quantities are equal because the maximum for the problem in the region $[i..i+k]$ for a fixed $L_i, L_p, L_{i+k}$ can be obtaining by finding $w(i,p,L_i,L_p)$ and $w(p,i+k,L_p,L_{i+k})$. We make this statement precise now. Let $L'_{i+1}, \ldots, L'_{p-1}$ be some codon sequences such that $L_i, L'_{i+1}, \ldots, L'_{p-1}, L_p$ satisfies $\mathrm{E}(G)||_{[i..p]}$, and $L''_{p+1}, \ldots, L''_{i+k-1}$ be a codon sequence such that $L_p, L''_{p+1}, \ldots, L''_{i+k-1}, L_{i+k}$ satisfies $\mathrm{E}(G)||_{[p..i+k]}$. We have to show that $L_i, L'_{i+1}, \ldots, L'_{p-1}, L_p, L''_{p+1}, \ldots, L''_{i+k-1}, L_{i+k}$ satisfies $\mathrm{E}(G)||_{[i..i+k]}$. $L_i$ and $L_{i+k}$ are compatible since we have assumed $(i, L_i) \equiv_{E(G)} (i+k, L_{i+k})$. The compatibility within regions $[i..p]$ and $[p..i+k]$ hold by assumption. The compatibility for vertex pairs $(u_r, u_s)$ with $r \in [i..p-1]$, and the second point is in $s \in [p+1..i+k]$ holds since $(i,p)$ is a maximal edge, and there are no edges in $\mathrm{E}(G^{\mathrm{impl}})$ between $r \in [i+1..p-1]$ and $s \in [p+1..i+k]$. □

## 3.1 Algorithm Based on the Recurrence

This theorem gives us a recursive algorithm to find the optimum value and an assignment of codons attaining that value for an input $(G, f_1, \ldots, f_n)$. We present the properties of the algorithm at the top-most level of

the recursion. Recall that our goal is to compute $\max_{L_1,L_n} w(1,n,L_1,L_n)$ and a corresponding assignment of codons. Let $p = \mathrm{next}(1,n)$. Since each vertex in $G^{\mathrm{impl}}$ has degree at most 3, $p$ can be found in at most 3 steps. If $w(1,p,L_1,L_p)$ and $w(p,n,L_p,L_n)$ is known for all choices of $L_1, L_p, L_n$, then we can compute $\max_{L_1,L_n} w(1,n,L_1,L_n)$. Observe that we can find a codon $L_p$ that achieves this maximum value. This computation takes constant time ($64^3 + 3$ to be exact). In particular, we can compute $w(1,n,L_1,L_n)$ for all choices of $L_1, L_n$ and a corresponding assignment to $L_p$.

Now we have to check how many subproblems will be generated. First, note that the subproblems, which are generated, are determined by the result of the $\mathrm{next}(\cdot,\cdot)$ function. Since $G^{\mathrm{impl}}$ is an outer-planar, the application of $\mathrm{next}(\cdot,\cdot)$ in some specific subproblem will always return a new position (i.e., a position that has not been considered in any other subproblem). Hence, we get only $n$ subproblems, and each one takes only at most $64^3 + 3$ time. Therefore, we can compute $\max_{L_1,L_n} w(1,n,L_1,L_n)$ and an assignment of codons attaining the optimum value in $(64^3 + 3)n$ time.

### 3.2 Results

We have implemented the program in the constraint programming language Oz [Smolka, 1995]. We have then used our program to search for proteins allowing selenocysteine incorporation in the complete genome of *E. coli*. Using our program, we have found modifications of several proteins that have high similarities to both the original amino-acid sequence, and to the SECIS-element, and allow to form the hairpin structure as required. These results are shown below in Table 1.

| name | PAM | orgAmino | modAmino | mRNA |
|---|---|---|---|---|
| yahE | 53 | RRWLSPTLQM | RRWLVPSLDL | CGACGAUGGUUGGUACCAAGUCUGGACCUA |
| mhpC | 56 | RIWLVKRQNR | RIWLVRRRDR | CGAAUAUGGUUGGUACGCAGGCGCGACCGA |
| ybdJ | 53 | LWFLVLGAIE | LWFLVPGPVE | CUAUGGUUCCUCGUACCGGGUCCGGUCGAG |
| ybiW | 54 | PWWRGQTVQD | PWWRVRSLDE | CCAUGGUGGCGCGUACGAAGUCUCGACGAG |
| ycaI | 54 | PEWQLPPVLR | PEWQLPSLVR | CCAGAAUGGCAGCUACCAAGUCUGGUGCGG |
| hyaF | 54 | GLWRVRRRRG | PLWRVRRRDG | CCAUUAUGGCGCGUACGCAGGCGCGACGGG |
| holB | 53 | RLHYLAPPPE | RLHYLASPPE | CGAUUACACUACCUAGCGAGUCCGCCGGAA |
| ymfO | 53 | QRWPEGDRRE | QRWPVGGRRE | CAACGAUGGCCCGUAGGCGGUCGCCGCGAG |
| hnr | 59 | QIWGTGGRLR | QIWGVGGRLR | CAAAUAUGGGGGGUAGGCGGUCGCCUCCGC |
| yciS | 53 | GLFWLRVRVS | PLFWLRSRVP | CCAUUAUUCUGGCUACGCAGUCGCGUGCCA |
| yeeP | 53 | HEWDMAGIQP | HEWELAGLQP | CACGAAUGGGAGCUAGCAGGUCUGCAGCCC |
| b2085 | 53 | RWYLMGEGEM | RWYLLGSPQL | CGAUGGUACUUGCUAGGGAGUCCCCAGCUA |
| b2710 | 53 | QWIYDPAKGE | QWIYVPSRGE | CAAUGGAUAUACGUACCCAGUCGGGGCGAA |
| ygcX | 54 | DWYRLRHEEA | QWYRLRRREA | CAAUGGUACCGCCUACGCAGGCGCGAGGCG |
| recC | 53 | RYYWGDIKDP | RYYWVPSRDP | CGAUACUACUGGGUACCCAGUCGGGACCCA |
| b3027 | 57 | GWWLFWGRFI | PWWLLRGRVV | CCAUGGUGGCUCCUACGCGGUCGCGUGGUG |
| yhcR | 54 | LFYLISRLFV | LFYLVARLLV | CUAUUCUACCUCGUAGCAAGGCUGCUCGUG |
| yrfG | 53 | LDYWSEQLGL | LDYWVPRLGL | CUAGACUACUGGGUACCAAGGCUGGGCCUA |
| yidL | 53 | SEAWLRRLFL | PEAWLRRLVL | CCAGAAGCAUGGCUACGAAGGCUCGUGCUA |
| yieN | 54 | LWYDAQSLNL | LWYEVRSLDL | CUAUGGUACGAGGUACGAAGUCUCGACCUC |
| yjiM | 53 | PYFYFSDLVV | PYFYLPGLVV | CCAUACUUCUACCUACCAGGUCUGGUGGUA |

Table 1: Some Results for *E. coli* genome

# 4 How hard is it to solve MRSO?

In the previous section, we have designed an efficient algorithm for $MRSO$ when $G^{\mathrm{impl}}$ is an outer-planar graph. This algorithm is suitable for our goal of selenocysteine insertion. In order to show the computational properties that makes the problem simple, and given that the $MRSO$ model attempts to capture more than selenocysteine insertion, we consider the case when $G^{\mathrm{impl}}$ is not necessarily outer-planar. We will just outline NP-hardness of this more general problem (for details, the reader is referred to [Backofen et al., ]). Note that our algorithm in the previous section depends heavily on the outer-planarity of the given graph.

We observe that when the graph is not restricted to be outer-planar, $MRSO$ is at least as hard as *3SAT* which is a special case of *boolean satisfiability* [Garey and Johnson, 1979]. To show this, we reduce an instance of the $3SAT$ problem to an instance of $MRSO$. It will follow easily that this reduc-

tion is a polynomial time reduction. Consequently, any polynomial time algorithm for $MRSO$ can be used, with an additional polynomial time effort, to solve $3SAT$. The $3SAT$ problem is defined as follows [Garey and Johnson, 1979]

**3-Satisfiability (3SAT)**

Instance: Collection $C = \{c_1, \ldots, c_m\}$ of clauses on a finite set $U$ of variables such that $|c_i| = 3$ for $1 \leq i \leq m$. For example, $(x_1 \vee x_2 \vee \neg x_3)$ is a clause whose variables are $x_1, x_2$, and $x_3$. These variables take values from the set $\{0, 1\}$ and the clause evaluates to true if at least one of its terms takes the value 1. $\neg x_3$ is the complement of $x_3$, that is if $\neg x_3 = 1 - x_3$.

Question: Is there a truth assignment of $U$ that satisfies all the clauses in $C$?

For example, $(x_1 \vee x_2 \vee \neg x_3)$ is a clause whose variables are $x_1, x_2$, and $x_3$. These variables take values from the set $\{0, 1\}$ and the clause evaluates to true if at least one of its terms takes the value 1. $\neg x_3$ is the complement of $x_3$, that is if $\neg x_3 = 1 - x_3$.

**Reduction** We present a simple example of the reduction as a representative of the general principles in the formal reduction. Let $\phi$ be $(\neg x_1 \vee x_2 \vee x_3) \wedge (x_4 \vee x_1 \vee \neg x_5)$. The variable $x_1$ is the only variable that has two different occurrences. Variables that occur more than once require a special treatment.

To construct an instance of $MRSO$ from $\phi$, first note that $\phi$ can be replaced equivalently by an extended formula, namely

$$\underbrace{(\neg y_1 \vee y_2 \vee y_3) \wedge (y_4 \vee y_5 \vee \neg y_6)}_{\phi_C} \quad \wedge \quad \underbrace{\mathrm{Equal}(y_1, y_5)}_{\phi_V}. \tag{1}$$

Here, $\mathrm{Equal}(x, y)$ is just a new boolean function expressing that $x$ and $y$ have the same valuation. The variables $y_1$ and $y_5$ (written in grey) are variables whose occurrences correspond to $x_1$. Thus, it is clear that their valuation must always be equal.

Now we will construct an instance of $MRSD$ from $\phi_C \wedge \phi_V$. We need to express the satisfiability of $\phi_C \wedge \phi_V$ by just using similarity functions and complementarity conditions. For this purpose, we use functions of the form $f : \{0, 1, 2, 3\}^3 \to \{0, 1\}$ as similarity functions (for simplicity we associate $\{0, 1, 2, 3\}$ with $\{A, T, C, G\}$; note that $A$ and $T$ are associated with $0, 1$, which implies that in this case, the boolean complement coincide with the nucleotide complement). Since the range of these functions is $\{0, 1\}$, we can combine them using Boolean operators. The complementarity condition will be expressed by introducing for every variable $y$ also a variable $y^C$, which will always have an assignment complementary to the assignment of $y$.

For $\phi_C$ as given in (1), we introduce functions $Cf_1(x, y, z)$ and $Cf_2(x, y, z)$ with

$$Cf_1(x, y, z) = \begin{cases} 0 & \text{if } x > 1 \text{ or } y > 1 \text{ or } z > 1 \\ 1 & \text{else if } \neg x \vee y \vee z \text{ is true} \\ 0 & \text{otherwise.} \end{cases}$$

$$Cf_2(x, y, z) = \begin{cases} 0 & \text{if } x > 1 \text{ or } y > 1 \text{ or } z > 1 \\ 1 & \text{else if } x \vee y \vee \neg z \text{ is true} \\ 0 & \text{otherwise.} \end{cases}$$

The satisfiability of $\phi_C$ is captured by assignments that evaluate $Cf_1(y_1, y_2, y_3) \wedge Cf_2(y_4, y_5, y_6)$ to 1.

For expressing $\mathrm{Equal}(y_1, y_5)$, we introduce additional variables $z_1$ and $z_2$, and a function $Ef : \{0, 1, 2, 3\}^3 \to \{0, 1\}$ with

$$Ef(x, y, z) = \begin{cases} 0 & \text{if } x > 1 \text{ or } y > 1 \text{ or } z > 1 \\ 1 & \text{else if } x = 0 \wedge y = z = 1 \\ 1 & \text{else if } x = 1 \wedge y = z = 0 \\ 0 & \text{otherwise.} \end{cases}$$

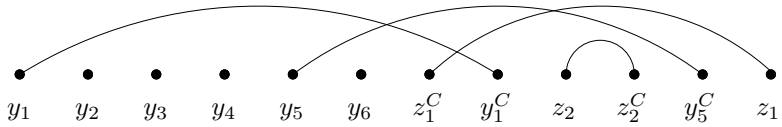
$Ef(x, y, z)$ guarantees that $z$ is equal to $y$, and that $x$ is the (boolean) complement of $y$. For $\mathrm{Equal}(y_1, y_5)$, it suffices to express $\mathrm{Equal}(y_1^C, y_5^C)$ for our purpose. Using $Ef(x, y, z)$, we can now express $\mathrm{Equal}(y_1^C, y_5^C)$ by

$$Ef(z_1^C, y_1^C, z_2) \wedge Ef(z_2^C, y_5^C, z_1).$$

The satisfiability of equation (1) is now captured by

$$Cf_1(y_1, y_2, y_3) \wedge Cf_2(y_4, y_5, y_6) \wedge Ef(z_1^C, y_1^C, z_2) \wedge Ef(z_2^C, y_5^C, z_1).$$

Thus, we have to construct and instance of $MRSO$ from this equation. Note that each variable or complemented variable occurs exactly once in the above formula. To complete, we define a graph $G$ on the vertices $\{y_1, y_2, y_3, y_4, y_5, y_6, z_1^C, y_1^C, z_2, z_2^C, y_5^C, z_1\}$, which connects the complementary variables. I.e., $G$ has the form




The similarity functions are just defined by

$$f_1 = Cf_1,\ f_2 = Cf_2,\ f_3 = Ef,\ f_4 = Ef.$$

Therefore, from $\phi$ we have constructed $(G, f_1, f_2, f_3, f_4)$ as an instance of MRSO. To complete the example, we observe that a satisfying assignment for $\phi$ yields a maximizing assignment to the instance of $MRSO$ obtained from $\phi$ and vice versa.

# 5 Acknowledgment

The first author likes to thank Prof. Böck from the Institute of Microbiology for explaining to him the biochemical background of selenocysteine, for many suggestions and the fruitful cooperation on the problems discussed in this paper. Furthermore, he likes to thank Prof. Clote for pointing out the problem, and for initiating the collaboration with Prof. Böck. He would also like to thank Sebastian Will for many discussion, and for reading draft versions of this paper. The authors thank the anonymous referees for their detailed comments.

# References

[Backofen et al., ] Backofen, R., Narayanaswamy, N., and F.Swidan. On the complexity of protien similarity search under mrna structure constraints. In *Proceedings of STACS 2002, LNCS 2258:274-286,2002*.

[Baron et al., 1993] Baron, C., Heider, J., and Böck, A. (1993). Interaction of translation factor SELB with the formate dehydrogenase H selenopolypeptide mRNA. *Proc. Natl. Acad. Sci. USA*, 90(9):4181–5.

[Böck et al., 1991] Böck, A., Forchhammer, K., Heider, J., and Baron, C. (1991). Selenoprotein synthesis: an expansion of the genetic code. *Trends Biochem Sci*, 16(12):463–467.

[Brown and Arthur, 2001] Brown, K. M. and Arthur, J. R. (2001). Selenium, selenoproteins and human health: a review. *Public Health Nutr*, 4(2B):593–9.

[Copeland et al., 2000] Copeland, P. R., Fletcher, J. E., Carlson, B. A., Hatfield, D. L., and Driscoll, D. M. (2000). A novel RNA binding protein, SBP2, is required for the translation of mammalian selenoprotein mRNAs. *EMBO J*, 19(2):306–14.

[Fagegaltier et al., 2000] Fagegaltier, D., Lescure, A., Walczak, R., Carbon, P., and Krol, A. (2000). Structural analysis of new local features in SECIS RNA hairpins. *Nucleic Acids Res*, 28(14):2679–89.

[Farabaugh, 1996] Farabaugh, P. J. (1996). Programmed translational frameshifting. *Microbiology and Molecular Biology Reviews*, 60(1):103–134.

[Forchhammer et al., 1989] Forchhammer, K., Leinfelder, W., and Böck, A. (1989). Identification of a novel translation factor necessary for the incorporation of selenocysteine into protein. *Nature*, 342(6248):453–6.

[Garey and Johnson, 1979] Garey, M. R. and Johnson, D. S. (1979). Computers and intractability: A guide to the theory of NP-completeness. *W.H.Freeman*.

[Giedroc et al., 2000] Giedroc, D. P., Theimer, C. A., and Nixon, P. L. (2000). Structure, stability and function of rna pseudoknots involved in stimulating ribosomal frameshifting. *Journal of Molecular Biology*, 298(2):167–186.

[Harary, 1972] Harary, F. (1972). Graph theory. *Addison Wesley*.

[Hazebrouck et al., 2000] Hazebrouck, S., Camoin, L., Faltin, Z., Strosberg, A. D., and Eshdat, Y. (2000). Substituting selenocysteine for catalytic cysteine 41 enhances enzymatic activity of plant phospholipid hydroperoxide glutathione peroxidase expressed in *escherichia coli*. *Journal of Biological Chemistry*, 275(37):28715–28721.

[Heider et al., 1992] Heider, J., Baron, C., and Böck, A. (1992). Coding from a distance: dissection of the mrna determinants required for the incorporation of selenocysteine into protein. *EMBO J*, 11(10):3759–66.

[Heider and Böck, 1992] Heider, J. and Böck, A. (1992). Targeted insertion of selenocysteine into the alpha subunit of formate dehydrogenase from methanobacterium formicicum. *J Bacteriol*, 174(3):659–63.

[Heider et al., 1991] Heider, J., Forchhammer, K., Sawers, G., and Böck, A. (1991). Interspecies compatibility of selenoprotein biosynthesis in enterobacteriaceae. *Arch Microbiol*, 155(3):221–8.

[Hilgenfeld et al., 1996] Hilgenfeld, R., Böck, A., and Wilting, R. (1996). Structural model for the selenocysteine-specific elongation factor SelB. *Biochimie*, 78(11-12):971–8.

[Hofacker et al., 1994] Hofacker, I., Fontana, W., Stadler, P., Bonhoeffer, S., Tacker, M., and Schuster, P. (1994). Fast folding and comparison of rna secondary structures. *Monatshefte f. Chemie*, 125:167–188.

[Hüttenhofer et al., 1996] Hüttenhofer, A., Westhof, E., and Böck, A. (1996). Solution structure of mRNA hairpins promoting selenocysteine incorporation in Escherichia coli and their base-specific interaction with special elongation factor SELB. *RNA*, 2(4):354–66.

[Klug et al., 1999] Klug, S. J., Hüttenhofer, A., and Famulok, M. (1999). In vitro selection of RNA aptamers that bind special elongation factor SelB, a protein with multiple RNA-binding sites, reveals one major interaction domain at the carboxyl terminus. *RNA*, 5:1180–1190.

[Klug et al., 1997] Klug, S. J., Hüttenhofer, A., Kraomayer, M., and Famulok, M. (1997). In vitro and in vivo characterization of novel mrna motifs that bind special elongation factor selb. *Proc. Natl. Acad. Sci. USA*, 94(13):6676–6681.

[Kromayer et al., 1999] Kromayer, M., Neuhierl, B., Friebel, A., and Böck, A. (1999). Genetic probing of the interaction between the translation factor SelB and its mRNA binding element in Escherichia coli. *Mol Gen Genet*, 262(4-5):800–6.

[Kromayer et al., 1996] Kromayer, M., Wilting, R., Tormay, P., and Böck, A. (1996). Domain structure of the prokaryotic selenocysteine-specific elongation factor SelB. *Journal of Molecular Biology*, 262(4):413–20.

[Kryukov et al., 1999] Kryukov, G. V., Kryukov, V. M., and Gladyshev, V. N. (1999). New mammalian selenocysteine-containing proteins identified with an algorithm that searches for selenocysteine insertion sequence elements. *J Biol Chem*, 274(48):33888–33897.

[Liu et al., 1998] Liu, Z., Reches, M., Groisman, I., and Engelberg-Kulka, H. (1998). The nature of the minimal 'selenocysteine insertion sequence' (secis) in escherichia coli. *Nucleic Acids Research*, 26(4):896–902.

[Low and Berry, 1996] Low, S. C. and Berry, M. J. (1996). Knowing when not to stop: selenocysteine incorporation in eukaryotes. *Trends in Biochemical Sciences*, 21(6):203–208.

[Rother et al., 2001] Rother, M., Resch, A., Wilting, R., and Bock, A. (2001). Selenoprotein synthesis in archaea. *Biofactors*, 14(1-4):75–83.

[Sawers et al., 1991] Sawers, G., Heider, J., Zehelein, E., and Böck, A. (1991). Expression and operon structure of the sel genes of Escherichia coli and identification of a third selenium-containing formate dehydrogenase isoenzyme. *J Bacteriol.*, 173(16):4983–93.

[Smolka, 1995] Smolka, G. (1995). The Oz programming model. In van Leeuwen, J., editor, *Computer Science Today*, Lecture Notes in Computer Science, vol. 1000, pages 324–343. Springer-Verlag, Berlin.

[Stadtman, 1996] Stadtman, T. C. (1996). Selenocysteine. *Annual review of biochemistry*, 65:83–100.

[Zinoni et al., 1990] Zinoni, F., Heider, J., and Böck, A. (1990). Features of the formate dehydrogenase mRNA necessary for decoding of the UGA codon as selenocysteine. *Proc. Natl. Acad. Sci. USA*, 87(12):4660–4.

# A Pseudocodes for Programs

```
Node(
   i: int
   j: int
   p: int
   values: array[64][64]
   trace: array[64][64]
   ipnode: Node
   pjnode: Node)
```

Figure 3: Node structure.

**Node data type:** The Node data type is used to compute $\max_{L_i,L_j} w(i,j,L_i,L_j)$. A variable of this data type has fields to store $i, j$, and $p = \text{next}(i,j)$. In addition, it stores two pointers $IPnode, PJnode$ to variables of Node data type which store information to compute $\max_{L_i,L_p} w(i,p,L_i,L_p)$ and $\max_{L_p,L_j} w(p,j,L_p,L_j)$, respectively. The value and trace fields are $64 \times 64$ matrices which are indexed by pairs of codons $(L_i, L_j)$. For each pair of codons $(L_i, L_j)$, $value[L_i][L_j] = w(i,j,L_i,L_j)$. Similarly, $trace[L_i][L_j]$ stores the first encountered codon $L_p$ such that $w(i,j,L_i,L_j) = w(i,p,L_i,L_p) + w(p,j,L_p,L_j) - f_p(L_p)$.

**Program** The input to the program is $n$ and an instance $(G, f_1, \ldots, f_n)$ of $MRSO$. The program proceeds by constructing a rooted binary tree whose vertices are variables of the Node data type. Each vertex has an interval associated with it. The root vertex corresponds to the interval $[1..n]$. A vertex is a leaf if the interval associated with it is of unit length. Otherwise, a vertex in the tree with an associated interval $[i..j]$ has two children: the vertex associated with $[i..p]$ (the left child) and the vertex associated with $[p..j]$ (the right child) where, $p = \text{next}(i,j)$. The two main subroutines of the program are $generate\_and\_fill$ and $calc\_trace\_rec$ (for the pseudocode of the program and its subroutine, see Appendix). The subroutine $generate\_and\_fill$ constructs the tree recursively starting with the root vertex and fills the value and trace entries of the leaf vertices it encounters. The value and trace matrices of the non-leaf vertices in the tree are filled by the subroutine $calc\_with\_sub$ in a bottom up manner (by a post order traversal of the tree). After the tree has been constructed, the maximum value of similarity is obtained in $main$ from the value matrix of the root vertex of the tree. A codon assignment that attains the maximum value is obtained by the subroutine $calc\_trace\_rec$. $calc\_trace\_rec$ is a recursive implementation of the inorder traversal of a tree and an optimum codon assignment is written into the global variable *assignment*.

**Variables:** The global varaible *assignment* is an array and contains an optimal soluton in the end.

**Program A.1** The main program

```
main(n, G, f_1, ..., f_n)
begin
  N = generate_and_fill(1,n)
  assignment = array[n]
  max = -∞    max_L1 = undef    max_Ln = undef
  for (L_1, L_n) = (1,1) to (64,64)
      if (N.values[L_1][L_n] > max) then
         max = N.values[L_1][L_n]
         max_L1 = L_1
         max_Ln = L_n
      endif
  end
  assignment[1] = max_L1; assignment[n] = max_Ln
  calc_trace_rec(N,max_L1,max_Ln)
end
```

**Program A.2** Subroutine generate_and_fill generates the tree of $w$ entries

```
subroutine generate_and_fill(i,j)
begin
N = new Node; N.i = i; N.j = j
  if (j = i+1) then
     N.p = -1
     N.values = new array[64][64]
     for (L_i, L_j) = (1,1) to (64,64)
         if (i, L_i) ≡_E(G) (j, L_j) then
            values[L_i][L_j] = f_i(L_i) + f_j(L_j)
         else
            values[L_i][L_j] = -∞
         endif
     end
  else
     N.p = next(i,j)
     N.ipnode = generate_and_fill(i,p)
     N.pjnode = generate_and_fill(p,j)
     (N.values,N.trace) = calc_with_sub(i,j,p,N.ipnode,N.pjnode)
  endif
  return(N)
end
```

**Program A.3** calc_with_sub fills value and trace array for a single node

```
subroutine calc_with_sub(i,j,p,IPnode,PJnode)
begin
  values = new array[64][64]; trace = new array[64][64]
  for (L_i,L_j) = (1,1) to (64,64)
     if (i,L_i) ≠_E(G) (j,L_j) then
        values[L_i][L_j] = −∞  trace[L_i][L_j] = -1
     else
        max=−∞
        maxL_p = undef
        for L_p = 1 to 64
           val =   IPnode.values[L_i][L_p] + PJnode.values[L_p][L_j]
                 - f_p(L_p)
           if (val > max) then
              max = val
              maxL_p = L_p
           endif
        end
        values[L_i][L_j] = max
        trace[L_i][L_j] = maxL_p
     endif
  end
return(values,trace)
end
```

**Program A.4** calc_trace_rec calculates the array *assignment*

```
subroutine calc_trace_rec(N,maxL_l,maxL_r)
begin
  p = N.p
  if (p = -1) then return
  else
        maxL_p = N.trace[L_l][L_r]
        assignment[p] = maxL_p
        calc_trace_rec(N.ipnode,maxL_1,maxL_p)
        calc_trace_rec(N.pjnode,maxL_p,maxL_n)
  endif
end
```