(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
29 December 2004 (29.12.2004)

PCT

(10) International Publication Number
WO 2004/113563 A2

(51) International Patent Classification[7]: C12Q 1/68

(21) International Application Number:
PCT/DK2004/000429

(22) International Filing Date: 18 June 2004 (18.06.2004)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:

PA 2003 00933	20 June 2003 (20.06.2003)	DK
PA 2003 01066	12 July 2003 (12.07.2003)	DK
PA 2004 00242	17 February 2004 (17.02.2004)	DK
PA 2004 00353	1 March 2004 (01.03.2004)	DK
60/549,346	2 March 2004 (02.03.2004)	US

(71) Applicant *(for all designated States except US)*: **EXIQON A/S** [DK/DK]; Bygstubben 9, DK-2950 Vedbæk (DK).

(72) Inventors; and

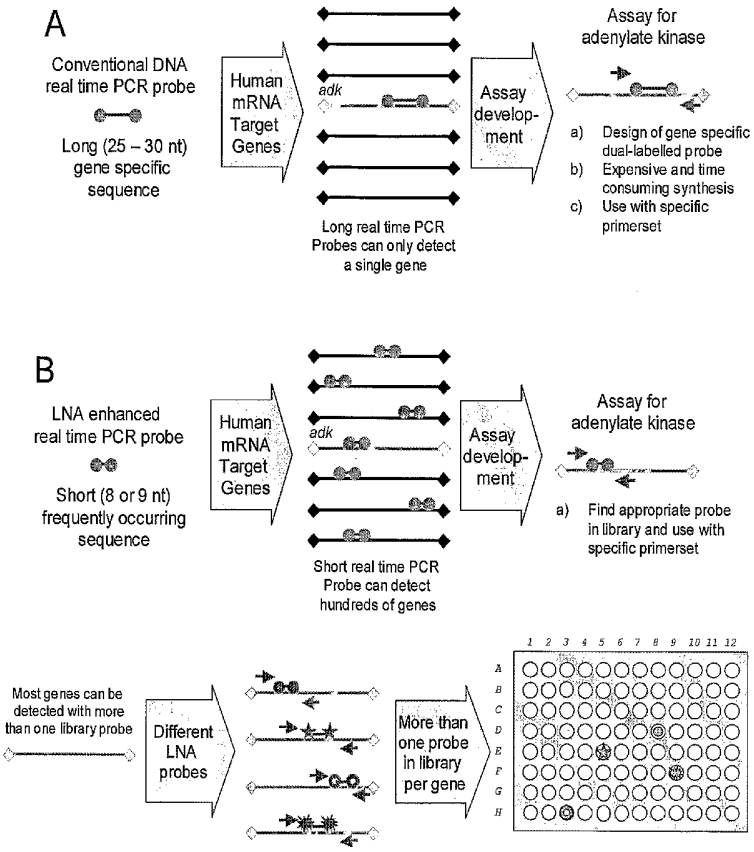
(75) Inventors/Applicants *(for US only)*: **RAMSING, Niels, B.** [DK/DK]; Ellebergvej 23, DK-8240 Risskov (DK). **MOURITZEN, Peter** [DK/DK]; Tranevej 12, DK-4040 Jyllinge (DK). **ECHWALD, Søren, Morgenthaler** [DK/DK]; Gammel Mejerivej 5, DK-3050 Humlebæk (DK). **TOLSTRUP, Niels** [DK/DK]; Bellevuevej 7, 1. th., DK-2930 Klampenborg (DK).

(74) Agent: **INSPICOS A/S**; Bøge Allé 5, P.O. Box 45, DK-2970 Hørsholm (DK).

(81) Designated States *(unless otherwise indicated, for every kind of national protection available)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BW, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NA, NI, NO, NZ, OM, PG, PH, PL, PT, RO, RU, SC, SD, SE, SG, SK, SL, SY, TJ, TM,

[Continued on next page]

(54) Title: PROBES, LIBRARIES AND KITS FOR ANALYSIS OF MIXTURES OF NUCLEIC ACIDS AND METHODS FOR CONSTRUCTING THE SAME



(57) Abstract: The invention relates to nucleic acid probes, nucleic acid probe libraries, and kits for detecting, classifying, or quantitating components in a complex mixture of nucleic acids, such as a transcriptome, and methods of using the same. The invention also relates to methods of identifying nucleic acid probes useful in the probe libraries and to methods of identifying a means for detection of a given nucleic acid.

WO 2004/113563 A2

TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, YU, ZA, ZM, ZW.

(84) Designated States *(unless otherwise indicated, for every kind of regional protection available)*: ARIPO (BW, GH, GM, KE, LS, MW, MZ, NA, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European (AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HU, IE, IT, LU, MC, NL, PL, PT, RO, SE, SI, SK, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

Published:

— *without international search report and to be republished upon receipt of that report*

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

PROBES, LIBRARIES AND KITS FOR ANALYSIS OF MIXTURES OF NUCLEIC ACIDS AND METHODS FOR CONSTRUCTING THE SAME

FIELD OF THE INVENTION

The invention relates to nucleic acid probes, nucleic acid probe libraries, and kits for detecting, classifying, or quantitating components in a complex mixture of nucleic acids, such as a transcriptome, and methods of using the same.

BACKGROUND OF THE INVENTION

With the advent of microarrays for profiling the expression of thousands of genes, such as GeneChip™ arrays (Affymetrix, Inc., Santa Clara, CA), correlations between expressed genes and cellular phenotypes may be identified at a fraction at the cost and labour necessary for traditional methods, such as Northern- or dot-blot analysis. Microarrays permit the development of multiple parallel assays for identifying and validating biomarkers of disease and drug targets which can be used in diagnosis and treatment. Gene expression profiles can also be used to estimate and predict metabolic and toxicological consequences of exposure to an agent (e.g., such as a drug, a potential toxin or carcinogen, etc.) or a condition (e.g., temperature, pH, etc).

Microarray experiments often yield redundant data, only a fraction of which has value for the experimenter. Additionally, because of the highly parallel format of microarray-based assays, conditions may not be optimal for individual capture probes. For these reasons, microarray experiments are most often followed up by, or sequentially replaced by, confirmatory studies using single-gene homogeneous assays. These are most often quantitative PCR-based methods such as the 5′ nuclease assay or other types of dual labelled probe quantitative assays. However, these assays are still time-consuming, single-reaction assays that are hampered by high costs and time-consuming probe design procedures. Further, 5′ nuclease assay probes are relatively large (e.g., 15-30 nucleotides). Thus, the limitations in homogeneous assay systems currently known create a bottleneck in the validation of microarray findings, and in focused target validation procedures.

An approach to avoid this bottleneck is to omit the expensive dual-labelled indicator probes used in 5′ nuclease assay procedures and molecular beacons and instead use non-sequence-specific DNA intercalating dyes such as SYBR Green that fluoresce upon binding to double-stranded but not single-stranded DNA. Using such dyes, it is possible to universally detect any amplified sequence in real-time. However, this technology is hampered by several

problems. For example, nonspecific priming during the PCR amplification process can generate unintentional non-target amplicons that will contribute in the quantification process. Further, interactions between PCR primers in the reaction to form "primer-dimers" are common. Due to the high concentration of primers typically used in a PCR reaction, this can lead to significant amounts of short double-stranded non-target amplicons that also bind intercalating dyes. Therefore, the preferred method of quantitating mRNA by real-time PCR uses sequence- specific detection probes.

One approach for avoiding the problem of random amplification and the formation of primer-dimers is to use generic detection probes that may be used to detect a large number of different types of nucleic acid molecules, while retaining some sequence specificity has been described by Simeonov, et al. (*Nucleic Acid Research 30(17)*: *91*, 2002; U.S. Patent Publication 20020197630) and involves the use of a library of probes comprising more than 10% of all possible sequences of a given length (or lengths). The library can include various non-natural nucleobases and other modifications to stabilize binding of probes/primers in the library to a target sequence. Even so, a minimal length of at least 8 bases is required for most sequences to attain a degree of stability that is compatible with most assay conditions relevant for applications such as real time PCR. Because a universal library of all possible 8-mers contains 65,536 different sequences, even the smallest library previously considered by Simeonov, et al. contains more than 10% of all possibilities, i.e. at least 6554 sequences which is impractical to handle and vastly expensive to construct.

From a practical point of view, several factors limit the ease of use and accessibility of contemporary homogeneous assays applications. The problems encountered by users of conventional assay technologies include:

- prohibitively high costs when attempting to detect many different genes in a few samples, because the price to purchase a probe for each transcript is high.
- the synthesis of labelled probes is time-consuming and often the time from order to receipt from manufacturer is more than 1 week.
- User-designed kits may not work the first time and validated kits are expensive per assay.
- it is difficult to quickly test for a new target or iteratively improve probe design.
- the exact probe sequence of commercial validated probes may be unknown for the customer resulting in problems with evaluation of results and suitability for scientific publication.
- When assay conditions or components are obscure it may be impossible to order reagents from alternative source.

The described invention address these practical problems and aim to ensure rapid and inexpensive assay development of accurate and specific assays for quantification of gene transcripts.

SUMMARY OF THE INVENTION

It is desirable to be able to quantify the expression of most genes (e.g., >98%) in e.g. the human transcriptome using a limited number of oligonucleotide detection probes in a homogeneous assay system. The present invention solves the problems faced by contemporary approaches to homogeneous assays outlined above by providing a method for construction of generic multi-probes with sufficient sequence specificity - so that they are unlikely to detect a randomly amplified sequence fragment or primer-dimers - but are still capable of detecting many different target sequences each. Such probes are usable in different assays and may be combined in small probe libraries (50 to 500 probes) that can be used to detect and/or quantify individual components in complex mixtures composed of thousands of different nucleic acids (e.g. detecting individual transcripts in the human transcriptome composed of >30,000 different nucleic acids.) when combined with a target specific primer set.

Each multi-probe comprises two elements: 1) a detection element or detection moiety consisting of one or more labels to detect the binding of the probe to the target; and 2) a recognition element or recognition sequence tag ensuring the binding to the specific target(s) of interest. The detection element can be any of a variety of detection principles used in homogeneous assays. The detection of binding is either direct by a measurable change in the properties of one or more of the labels following binding to the target (e.g. a molecular beacon type assay with or without stem structure) or indirect by a subsequent reaction following binding (e.g. cleavage by the 5′ nuclease activity of the DNA polymerase in 5′ nuclease assays).

The recognition element is a novel component of the present invention. It comprises a short oligonucleotide moiety whose sequence has been selected to enable detection of a large subset of target nucleotides in a given complex sample mixture. The novel probes designed to detect many different target molecules each are referred to as multi-probes. The concept of designing a probe for multiple targets and exploit the recurrence of a short recognition sequence by selecting the most frequently encountered sequences is novel and contrary to conventional probes that are designed to be as specific as possible for a single target sequence. The surrounding primers and the choice of probe sequence in combination subsequently ensures the specificity of the multi-probes. The novel design principles arising from attempts to address the largest number of targets with the smallest number of probes are

likewise part of the invention. This is enabled by the discovery that very short 8-9 mer LNA mix-mer probes are compatible with PCR based assays. In one aspect of the present invention modified or analogue nucleobases, nucleosidic bases or nucleotides are incorporated in the recognition element, possibly together with minor groove binders and other modifications, that all aim to stabilize the duplex formed between the probe and the target molecule so that the shortest possible probe sequence with the widest range of targets can be used. In a preferred aspect of the invention the modifications are incorporation of LNA residues to reduce the length of the recognition element to 8 or 9 nucleotides while maintaining sufficient stability of the formed duplex to be detectable under ordinary assay conditions.

Preferably, the multi-probes are modified in order to increase the binding affinity of the probe for a target sequence by at least two-fold compared to a probe of the same sequence without the modification, under the same conditions for detection, e.g., such as PCR conditions, or stringent hybridization conditions. The preferred modifications include, but are not limited to, inclusion of nucleobases, nucleosidic bases or nucleotides that has been modified by a chemical moiety or replaced by an analogue (e.g. including a ribose or deoxyribose analogue) or by unsing internucleotide linkages other than phosphodiester linkages (such as non-phosphate internucleotide linkages), all to increase the binding affinity. The preferred modifications may also include attachment of duplex stabilizing agents e.g., such as minor-groove-binders (MGB) or intercalating nucleic acids (INA). Additionally the preferred modifications may also include addition of non-discriminatory bases e.g., such as 5-nitroindole, which are capable of stabilizing duplex formation regardless of the nucleobase at the opposing position on the target strand. Finally, multi-probes composed of a non-sugar-phosphate backbone, e.g. such as PNA, that are capable of binding sequence specifically to a target sequence are also considered as modification. All the different binding affinity increased modifications mentioned above will in the following be referred to as "the stabilizing modification(s)", and the ensuing multi-probe will in the following also be referred to as "modified oligonucleotide". More preferably the binding affinity of the modified oligonucleotide is at least about 3-fold, 4-fold, 5-fold, or 20-fold higher than the binding of a probe of the same sequence but without the stabilizing modification(s).

Most preferably, the stabilizing modification(s) is inclusion of one or more LNA nucleotide analogs. Probes of from 6 to 12 nucleotides according to the invention may comprise from 1 to 8 stabilizing nucleotides, such as LNA nucleotides. When at least two LNA nucleotides are included, these may be consecutive or separated by one or more non-LNA nucleotides. In one aspect, LNA nucleotides are alpha and/or xylo LNA nucleotides.

The invention also provides oligomer multi-probe library useful under conditions used in NASBA based assays.

NASBA is a specific, isothermal method of nucleic acid amplification suited for the amplification of RNA. Nucleic acid isolation is achieved via lysis with guanidine thiocyanate plus Triton X-100 and ending with purified nucleic acid being eluted from silicon dioxide particles. Amplification by NASBA involves the coordinated activities of three enzymes, AMV Reverse Transcriptase, RNase H, and T7 RNA Polymerase. Quantitative detection is achieved by way of internal calibrators, added at isolation, which are co-amplified and subsequently identified along with the wild type of RNA using electro chemiluminescence.

The invention also provides an oligomer multi-probe library comprising multi-probes comprising at least one with stabilizing modifications as defined above. Preferably, the probes are less than about 20 nucleotides in length and more preferably less than 12 nucleotides, and most preferably about 8 or 9 nucleotides. Also, preferably, the library comprises less than about 3000 probes and more preferably the library comprises less than 500 probes and most preferably about 100 probes. The libraries containing labelled multi-probes may be used in a variety of applications depending on the type of detection element attached to the recognition element. These applications include, but are not limited to, dual or single labelled assays such as 5' nuclease assay, molecular beacon applications (see, e.g., Tyagi and Kramer Nat. Biotechnol. 14: 303-308, 1996) and other FRET-based assays.

In one aspect of the invention the multi-probes described above, are designed together to complement each other as a predefined subset of all possible sequences of the given lengths selected to be able to detect/characterize/quantify the largest number of nucleic acids in a complex mixture using the smallest number of multi-probe sequences. These predesigned small subsets of all possible sequences constitute a multi-probe library. The multi-probe libraries described by the present invention attains this functionality at a greatly reduced complexity by deliberately selecting the most commonly occurring oligomers of a given length or lengths while attempting to diversify the selection to get the best possible coverage of the complex nucleic acid target population. In one preferred aspect, probes of the library hybridize with more than about 60% of a target population of nucleic acids, such as a population of human mRNAs. More preferably, the probes hybridize with greater than 70%, greater than 80%, greater than 90%, greater than 95% and even greater than 98% of all target nucleic acid molecules in a population of target molecules (see, e.g., Fig. 1).

In a most preferred aspect of the invention, a probe library (i.e. such as about 100 multi-probes) comprising about 0.1 % of all possible sequences of the selected probe length(s), is capable of detecting, classifying, and/or quantifying more than 98% of mRNA transcripts in the transcriptome of any specific species, particulary mammals and more particular humans (i.e., > 35,000 different mRNA sequences). In fact, it is preferred that at least 85% of all

target nucleic acids in a target population are covered by a multi-probe library of the invention.

The problems with existing homogeneous assays mentioned above are addressed by the use of a multi-probe library according to the invention consisting of a minimal set of short detection probes selected so as to recognize or detect a majority of all expressed genes in a given cell type from a given organism. In one aspect, the library comprises probes that detect each transcript in a transcriptome of greater than about 10,000 genes, greater than about 15,000 genes, greater than about 20,000 genes, greater than about 25,000 genes, greater than about 30,000 genes or greater than about 35,000 genes or equivalent numbers of different mRNA transcripts. In one preferred aspect, the library comprises probes that detect mammalian transcripts sequences, e.g., such as mouse, rat, rabbit, monkey, or human sequences.

By providing a cost efficient multi-probe set useful for rapid development of quantitative real-time and end-point PCR assays, the present invention overcomes the limitations discussed above for contemporary homogeneous assays. The detection element of the multi-probes according to the invention may be single or doubly labelled (e.g. by comprising a label at each end of the probe, or an internal position). Thus, probes according to the invention can be adapted for use in 5' nuclease assays, molecular beacon assays, FRET assays, and other similar assays. In one aspect, the detection multi-probe comprises two labels capable of interacting with each other to produce a signal or to modify a signal, such that a signal or a change in a signal may be detected when the probe hybridizes to a target sequence. A particular aspect is when the two labels comprise a quencher and a reporter molecule.

In another aspect, the probe comprises a target-specific recognition segment capable of specifically hybridizing to a plurality of different nucleic acid molecules comprising the complementary recognition sequence. A particular detection aspect of the invention referred to as a "molecular beacon with a stem region" is when the recognition segment is flanked by first and second complementary hairpin-forming sequences which may anneal to form a hairpin. A reporter label is attached to the end of one complementary sequence and a quenching moiety is attached to the end of the other complementary sequence. The stem formed when the first and second complementary sequences are hybridized (i.e., when the probe recognition segment is not hybridized to its target) keeps these two labels in close proximity to each other, causing a signal produced by the reporter to be quenched by fluorescence resonance energy transfer (FRET). The proximity of the two labels is reduced when the probe is hybridized to a target sequence and the change in proximity produces a change in the interaction between the labels. Hybridization of the probe thus results in a signal (e.g. fluorescence) being produced by the reporter molecule, which can be detected and/or quantified.

In another aspect, the multi-probe comprises a reporter and a quencher molecule at opposing ends of the short recognition sequence, so that these moieties are in sufficient proximity to each other, that the quencher substantially reduces the signal produced by the reporter molecule. This is the case both when the probe is free in solution as well as when it is bound to the target nucleic acid. A particular detection aspect of the invention referred to as a "5' nuclease assay" is when the multi-probe may be susceptible to cleavage by the 5' nuclease activity of the DNA polymerase. This reaction may possibly result in separation of the quencher molecule from the reporter molecule and the production of a detectable signal. Thus, such probes can be used in amplification-based assays to detect and/or quantify the amplification process for a target nucleic acid.

In a first aspect, the present invention relates to libraries of multi-probes as discussed above. In such a library of oligonucleotide probes, each probe comprises a detection element and a recognition segment having a length of about x nucleotides, where some or all of the nucleobases in said oligonucleotides are substituted by non-natural bases having the effect of increasing binding affinity compared to natural nucleobases, and/or some or all of the nucleotide units of the oligonucleotide probe are modified with a chemical moiety to increase binding affinity, and/or where said oligonucleotides are modified with a chemical moiety to increase binding affinity, such that the probe has sufficient stability for binding to the target sequence under conditions suitable for detection, and wherein the number of different recognition segments comprises less than 10% of all possible segments of the given length, and wherein more than 90% of the probes can detect more than one complementary target in a target population of nucleic acids such that the library of oligonucleotide probes can detect a substantial fraction of all target sequences in a target population of nucleic acids.

The invention therefore relates to a library of oligonucleotide probes wherein each probe in the library consists of a recognition sequence tag and a detection moiety wherein at least one monomer in each oligonucleotide probe is a modified monomer analogue, increasing the binding affinity for the complementary target sequence relative to the corresponding unmodified oligodeoxyribonucleotide, such that the library probes have sufficient stability for sequence-specific binding and detection of a substantial fraction of a target nucleic acid in any given target population and wherein the number of different recognition sequences comprises less than 10% of all possible sequence tags of a given length(s).

The invention further relates to a library of oligonucleotide probes wherein the recognition sequence tag segment of the probes in the library have been modified in at least one of the following ways:

i) substitution with at least one non-naturally occurring nucleotide; and

ii) substitution with at least one chemical moiety to increase the stability of the probe.

Further, the invention relates to a library of oligonucleotide probes wherein the recognition sequence tag has a length of 6 to 12 nucleotides (*i.e.* 6, 7, 8, 9, 10, 11 or 12), and wherein the preferred length is 8 or 9 nucleotides.

Further, the invention relates to recognition sequence tags that are substituted with LNA nucleotides.

Moreover, the inventon relates to libraries of the invention where more than 90% of the oligonucleotide probes can bind and detect at least two target sequences in a nucleic acid population, preferably because the bound target sequences that are complementary to the recognition sequence of the probes.

Also preferably, the probe is capable of detecting more than one target in a target population of nucleic acids, e.g., the probe is capable of hybridizing to a plurality of different nucleic acid molecules contained within the target population of nucleic acids.

The invention also provides a method, system and computer program embedded in a computer readable medium ("a computer program product") for designing multi-probes comprising at least one stabilizing nucleobase. The method comprises querying a database of target sequences (e.g., such as a database of expressed sequences) and designing a small set of probes (e.g. such as 50 or 100 or 200 or 300 or 500) which: i) has sufficient binding stability to bind their respective target sequence under PCR conditions, ii) have limited propensity to form duplex structures with itself, and iii) are capable of binding to and detecting/quantifying at least about 60%, at least about 70%, at least about 80%, at least about 90% or at least about 95% of all the sequences in the given database of sequences, such as a database of expressed sequences.

Probes are designed *in silico*, which comprise all possible combinations of nucleotides of a given length forming a database of virtual candidate probes. These virtual probes are queried against the database of target sequences to identify probes that comprise the maximal ability to detect the most different target sequences in the database ("optimal probes"). Optimal probes so identified are removed from the virtual probe database. Additionally, target nucleic acids, which were identified by the previous set of optimal probes, are subtracted from the target nucleic acid database. The remaining probes are then queried against the remaining target sequences to identify a second set of optimal probes. The process is repeated until a set of probes is identified which can provide the desired coverage of the target sequence database. The set may be stored in a database as a source of sequences for transcriptome analysis. Multi-probes may be synthesized having recognition sequences, which correspond to those in the database to generate a library of multi-probes.

In one preferred aspect, the target sequence database comprises nucleic acid sequences corresponding to human mRNA (e.g., mRNA molecules, cDNAs, and the like).

In another aspect, the method further comprises calculating stability based on the assumption that the recognition sequence comprises at least one stabilizing nucleotide, such as an LNA molecule. In one preferred aspect the calculated stability is used to eliminate probe recognition sequences with inadequate stability from the database of virtual candidate probes prior to the initial query against the database of target sequence to initiate the identification of optimal probe recognition sequences.

In another aspect, the method further comprises calculating the propensity for a given probe recognition sequence to form a duplex structure with itself based on the assumption that the recognition sequence comprises at least one stabilizing nucleotide, such as an LNA molecule. In one preferred aspect the calculated propensity is used to eliminate probe recognition sequences that are likely to form probe duplexes from the database of virtual candidate probes prior to the initial query against the database of target sequence to initiate the determination of optimal probe recognition sequences.

In another aspect, the method further comprises evaluating the general applicability of a given candidate probe recognition sequence for inclusion in the growing set of optimal probe candidates by both a query against the remaining target sequences as well as a query against the original set of target sequences. In one preferred aspect only probe recognition sequences that are frequently found in both the remaining target sequences and in the original target sequences are added to in the growing set of optimal probe recognition sequences. In a most preferred aspect this is accomplished by calculating the product of the scores from these queries and selecting the probes recognition sequence with the highest product that still is among the probe recognition sequences with 20% best score in the query against the current targets.

The invention also provides a computer program embedded in a computer readable medium comprising instructions for searching a database comprising a plurality of different target sequences and for identifying a set of probe recognition sequences capable of identifying to at least about 60%, about 70%, about 80%, about 90% and about 95% of the sequences within the database. In one aspect, the program provides instructions for executing the method described above. In another aspect, the program provides instructions for implementing an algorithm as shown in Fig. 2. The invention further provides a system wherein the system comprises a memory for storing a database comprising sequence information for a plurality of different target sequences and also comprises an application program for executing the program instructions for searching the database for a set of probe recognition se-

quences which is capable of hybridizing to at least about 60%, about 70%, about 80%, about 90% and about 95% of the sequences within the database.

Another aspect of the invention relates to an oligonucleotide probe comprising a detection element and a recognition segment each independently having a length of about 1 to 8 nucleotides, wherein some or all of the nucleotides in the oligonucleotides are substituted by non-natural bases or base analogues having the effect of increasing binding affinity compared to natural nucleobases and/or some or all of the nucleotide units of the oligonucleotide probe are modified with a chemical moiety or replaced by an analogue to increase binding affinity, and/or where said oligonucleotides are modified with a chemical moiety or is an oligonucleotide analogue to increase binding affinity, such that the probe has sufficient stability for binding to the target sequence under conditions suitable for detection, and wherein the probe is capable of detecting more than one complementary target in a target population of nucleic acids.

A preferred embodiment of the invention is a kit for the characterization or detection or quantification of target nucleic acids comprising samples of a library of multi-probes. In one aspect, the kit comprises *in silico* protocols for their use. In another aspect, the kit comprises information relating to suggestions for obtaining inexpensive DNA primers. The probes contained within these kits may have any or all of the characteristics described above. In one preferred aspect, a plurality of probes comprises at least one stabilizing nucleotide, such as an LNA nucleotide. In another aspect, the plurality of probes comprises a nucleotide coupled to or stably associated with at least one chemical moiety for increasing the stability of binding of the probe. In a further preferred aspect, the kit comprises about 100 different probes. The kits according to the invention allow a user to quickly and efficiently develop an assay for thousands of different nucleic acid targets.

The invention further provides a multi-probe comprising one or more LNA nucleotide, which has a reduced length of about 8, or 9 nucleotides. By selecting commonly occurring 8 and 9-mers as targets it is possible to detect many different genes with the same probe. Each 8 or 9-mer probe can be used to detect more than 7000 different human mRNA sequences. The necessary specificity is then ensured by the combined effect of inexpensive DNA primers for the target gene and by the 8 or 9-mer probe sequence targeting the amplified DNA (Fig. 1).

In a preferred embodiment the present invention relates to an oligonucleotide multi-probe library comprising LNA-substituted octamers and nonamers of less than about 1000 sequences, preferably less than about 500 sequences, or more preferably less than about 200 sequences, such as consisting of about 100 different sequences selected so that the library is

able to recognize more than about 90%, more preferably more than about 95% and more preferably more than about 98% of mRNA sequences of a target organism or target organ.

Positive control samples:

A recurring problem in designing real-time PCR detection assays for multiple genes is that the success-rate of these de-novo designs is less than 100%. Troubleshooting a nonfunctional assay can be cumbersome since ideally, a target specific template is needed for each probe, to test the functionality of the detection probe. Furthermore, a target specific template can be useful as a positive control if it is unknown whether the target is available in the test sample. When operating with a limited number of detection probes in a probelibrary kit as described in the present invention (eg. 90), it is feasible to also provide positive control targets in the form of PCR-amplifiable templates containing all possible targets for the limited number of probes (eg. 90). This feature allows users to evaluate the function of each probe, and is not feasible for non-recurring probe-based assays, and thus constitutes a further beneficial feature of the invention. For the suggested preferred probe recognition sequences listed in Fig. 13, we have designed concatamers of control sequences for all probes, containing a PCR-amplifiable target for every probe in the 40 first probes.

Probe sequence selection

An important aspect of the present invention is the selection of optimal probe target sequences in order to target as many targets with as few probes as possible, given a target selection criteria. This may be achieved by deliberately selecting target sequences that occur more frequently than what would have been expected from a random distribution.

The invention therefore relates in one aspect to a method of selecting oligonucleotide sequences useful in a multi-probe library of the invention, the method comprising
a) providing a first list of all possible oligonucleotides of a predefined number of nucleotides, N (typically an integer selected from 6, 7, 8, 9, 10, 11, and 12, preferably 8 or 9), said oligonucleotides having a melting temperature, Tm, of at least 50°C (preferably at least 60°C),
b) providing a second list of target nucleic acid sequences (such as a list of a target nucleic acid population discussed herein),
c) identifying and storing for each member of said first list, the number of members from said second list, which include a sequence complementary to said each member,
d) selecting a member of said first list, which in the identification in step c matches the maximum number, identified in step c, of members from said second list,
e) adding the member selected in step d to a third list consisting of the selected oligonucleotides useful in the library according to any one of the preceding claims,

f) subtracting the member selected in step d from said first list to provide a revised first list,
m) repeating steps d through f until said third list consists of members which together will be contemplary to at least 30% of the members on the list of target nucleic acid sequences from step b (normally the percentage will be higher, such as at least 40%, at least 50%, at least 60%, at least 70%, at least 75%, at least 80%, at least 85%, at least 90%, at least 95%, or even higher such as at least 97%, at least 98% and even as high as at least 99%)..

It is preferred that the first list only includes oligonucleotides incapable of self-hybridization in order to render a subsequent use of the probes less prone to false positives.

The selection method may include a number of steps after step f, but before step m
g) subtraction of all members from said second list which include a sequence complementary to the member selected in step d to obtain a revised second list,
h) identification and storing of, for each member of said revised first list, the number of members from said revised second list, which include a sequence complementary to said each member, i) selecting a member of said first list, which in the identification in step h matches the maximum number, identified in step h, of members from said second list, or selecting a member of said first list provides the maximum number obtained by multiplying the number identified in step h with the number identified in step c,
j) addition of the member selected in step i to said third list,
k) subtraction of the member selected in step i from said revised first list, and
l) subtraction of all members from said revised second list which include a sequence or complementary to the member selected in step i.

The selection in step d after step c is conveniently preceded by identification of those members of said first list which hybridizes to more than a selected percentage (60% or higher such as the preferred 80%) of the maximum number of members from said second list so that only those members so identified are subjected to the selection in step d.

In the practical implementation of the selection method, said first, second and third lists are stored in the memory of a computer system, preferably in a database. The memory (also termed "computer readable medium") can be both volatile and non-volatile, *i.e.* any memory device conventionally used in computer systems: a random access memory (RAM), a read-only memory (ROM), a data storage device such as a hard disk, a CD-ROM, DVD-ROM, and any other known memory device.

The invention also provides a computer program product providing instructions for implementing the selection method, embedded in a computer-readable medium (defined as above). That is, the computer program may be compiled and loaded in an active computer

memory, or it may be loaded on a non-volatile storage device (optionally in a compressed format) from where it can be executed. Consequently, the invention also includes a system comprising a database of target sequences and an application program for executing the computer program. A source code for such a computer program is set forth in Fig. 17.

In a randomly distributed nucleic acid population, the occurrence of selected sequences of a given length will follow a statistical distribution defined by:

N1 = the complete length of the given nucleic acid population (eg. 76.002.917 base pairs as in the 1 June 30, 2003 release of RefSeq).

N2= the number of fragments comprising the nucleic acid population (eg 38.556 genes in the 1 June 30, 2003 release of RefSeq).

N3 = the length of the recognition sequence (eg. 9 base pairs)

N4 = the occurrence frequency

$N4 = (N1-((N3-1) \times 2 \times N2))/(4^{N3})$

Eg.

$\frac{76,002,917 - 8 \times 2 \times 38,556}{4^9}$ = approximately 287 occurrences of 9-mer sequences or

or

$\frac{76,002,917 - 7 \times 2 \times 38,556}{4^8}$ = approximately 1,151 occurrences of 8-mer sequences

Hence, as described in the example given above, a random 8-mer and 9-mer sequence would on average occur 1,151 and 287 times, respectively, in a random population of the described 38,556 mRNA sequences.

In the example above, the 76.002.917 base pairs originating from 38.556 genes would correspond to an average transcript length of 1971 bp, containing each 1971-16 or 1955 9-mer target sequences each. Thus as a statistical minimum, 38.556/1955/287 or 5671 9-mer probes would be needed for one probe to target each gene.

However, the occurrence of 9-mer sequences is not randomly distributed. In fact, a small subset of sequences occur at surprisingly high prevalence, up to over 30 times the prevalence anticipated from a random distribution. In a specific target population selected according to preferred criteria, preferably the most common sequences should be selected to increase the coverage of a selected library of probe target sequences. As described previously, selection should be step-wise, such that the selection of the most common target sequences is evaluated as well in the starting target population as well as in the population remaining after each selection step.

In a preferred embodiment of the invention the targets for the probelibrary are the entire expressed transcriptome.

Because the success rate of the reverse transcriptase reaction diminishes with the distance from the RT-primer used, and since using a poly-T primer targeting the poly-A tract in mRNAs is common, the above-mentioned target can further be restricted to only include the 1000 most proximal bases in each mRNA. This may result in the selection of another set of optimal probe target sequences for optimal coverage.

Likewise the above-mentioned target may be restricted to include only the 50 bp of coding region sequence flanking the introns of a gene to ensure assays that preferably only monitor mRNA and not genomic DNA or to only include regions not containing di-, tri- or tetra repeat sequences, to avoid repetitive binding or probes or primers or regions not containing know allelic variation, to avoid primer or probe mis-annealing due to sequence variations in target sequences or regions of extremely high GC-content to avoid inhibition of PCR amplification.

Depending on each target selection the optimal set of probes may vary, depending in the prevalence of target sequences in each target selection.

Selection of detection means and identification of single nucleic acids

Another part of the invention relates to identification of a means for detection of a target nucleic acid, the method comprising

A) inputting, into a computer system, data that uniquely identifies the nucleic acid sequence of said target nucleic acid, wherein said computer system comprises a database holding information of the composition of at least one library of nucleic acid probes of the invention, and wherein the computer system further comprises a database of target nucleic acid sequences for each probe of said at least one library and/or further comprises means for acquiring and comparing nucleic acid sequence data,

B) identifying, in the computer system, a probe from the at least one library, wherein the

sequence of the probe exists in the target nucleic acid sequence or a sequence complementary to the target nucleic acid sequence,

C) identifying, in the computer system, primer that will amplify the target nucleic acid sequence, and

D) providing, as identification of the specific means for detection, an output that points out the probe identified in step B and the sequences of the primers identified in step C.

The above-outlined method has several advantages in the event it is desired to rapidly and specifically identify a particular nucleic acid. If the researcher already has acquired a suitable multi-probe library of the invention, the method makes it possible within seconds to acquire information relating to which of the probes in the library one should use for a subsequent assay, *and* of the primers one should synthesize. The time factor is important, since synthesis of a primer pair can be accomplished overnight, whereas synthesis of the probe would normally be quite time-consuming and cumbersome.

To facilitate use of the method, the probe library can be identified (e.g. by means of a product code which essentially tells the computer system how the probe library is composed). Step A then comprises inputting, into the computer system, data that identifies the at least one library of nucleic acids from which it is desired to select a member for use in the specific means for detection.

The preferred inputting interface is an internet-based web-interface, because the method is conveniently stored on a web server to allow access from users who have acquired a probe library of the present invention. However, the method also would be useful as part of a installable computer application, which could be installed on a single computer or on a local area network.

In preferred embomiments of this method, the primers identified in step C are chosen so as to minimize the chance of amplifying genomic nucleic acids in a PCR reaction. This is of course only relevant where the sample is likely to contain genomic material. One simple way to minimize the chance of amplification of genomic nucleic acids is to include, in at least one of the primers, a nucleotide sequence which in genomic DNA is interrupted by an intron. In this way, the primer will only prime amplification of transcripts where the intron has been spliced out.

A further optimization of the method is to choose the primers in step C so as to minimize the length of amplicons obtained from PCR performed on the target nucleic acid sequence and it is furter also preferred to select the primers so as to optimize the GC content for performing a subsequent PCR.

As for the probe selection method, the selection method for detection means can be provided to the end-user as a computer program product providing instructions for implementing the method, embedded in a computer-readable medium. Consequently, the invention also provides for a system comprising a database of nucleic acid probes of the invention and an application program for executing this computer program.

The method and the computer programs and system allows for quantitative or qualitative determination of the presence of a target nucleic acid in a sample, comprising
i) identifying, by means of the detection means selection method of the invention, a specific means for detection of the target nucleic acid, where the specific means for detection comprises an oligonucleotide probe and a set of primers,
ii) obtaining the primers and the oligonucleotide probe identified in step i),
iii) subjecting the sample to a molecular amplification procedure in the presence of the primers and the oligonucleotide probe from step ii), and
iv) determining the presence of the target nucleic acid based on the outcome of step iii).

Conveniently, primers obtained in step ii) are obtained by synthesis and it is preferred that the oligonucleotide probe is obtained from a library of the present invention.

The molecular amplification method is typically a PCR or a NASBA procedure, but any *in vitro* method for specific amplification (and, possibly, detection) of a nucleic acid is useful. The preferred PCR procedure is a qPCR (also known as real-time reverse transcription PCR or kinetic RT-PCR).

Other aspects of the invention are discussed *infra*.

BRIEF DESCRIPTION OF THE DRAWINGS

Fig. 1 illustrates the use of conventional long probes in panel (A) as well as the properties and use of short multi-probes (B) from a library constructed according to the invention. The short multi-probes comprise a recognition segment chosen so that each probe sequence may be used to detect and/or quantify several different target sequences comprising the complementary recognition sequence. Fig. 1A shows a method according to the prior art. Fig. 1B shows a method according to one aspect of the invention.

Fig. 2 is a flow chart showing a method for designing multi-probe sequences for a library according to one aspect of the invention. The method can be implemented by executing instructions provided by a computer program embedded in a computer readable medium. In

one aspect, the program instructions are executed by a system, which comprises a database of sequences such as expressed sequences.

Fig. 3 is a graph illustrating the redundancy of probes targeting each gene within a 100-probe library according to one aspect of the invention. The y-axis shows the number of genes in the human transcriptome that are targeted by different number of probes in the library. It is apparent that a majority of all genes are targeted by several probes. The average number of probes per gene is 17.4.

Fig. 4 shows the theoretical coverage of the human transcriptome by a selection of hyper-abundant oligonucleotides of a given length. The graphs show the percentage of approximately 38.000 human mRNA sequences that can be detected by an increasing number of well-chosen short multi-probes of different length. The graph illustrates the theoretical coverage of the human transcriptome by optimally chosen (i.e. hyper-abundant, non-self complementary and thermally stable) short multi-probes of different lengths. The Homo sapiens transcriptome sequence was obtained from European Bioinformatics Institute (EMBL-EBI). A region of 1000 nt proximal to the 3' end of each mRNA sequence was used for the analysis (from 50 nt to 1050 nt upstream from the 3' end). As the amplification of each sequence is by PCR both strands of the amplified duplex was considered a valid target for multi-probes in the probe library. Probe sequences that even with LNA substitutions have inadequate Tm, as well as self-complementary probe sequences are excluded.

Fig. 5 shows the MALDI-MS spectrum of the oligonuclotide probe EQ13992, showing $[M-H]^-$ = 4121,3 Da.

Fig. 6 shows representative real time PCR curves for 9-mer multi-probes detecting target sequences in a dual labelled probe assay. Results are from real time PCR reactions with 9 nt long LNA enhanced dual labelled probes targeting different 9-mer sequences within the same gene. Each of the three different dual labelled probes were analysed in PCRs generating either the 469, the 570 or the 671 SSA4 amplicons (each between 81 to 95 nt long). Dual labelled probe 469, 570, and 671 is shown in Panel a, b, and c, respectively. Each probe only detects the amplicon it was designed to detect. The C_t values were 23.7, 23.2, and 23.4 for the dual labelled probes 469, 570, and 671, respectively. 2×10^7 copies of the SSA4 cDNA were added as template. The high similarity between results despite differences in both probe sequences and their individual primer pairs indicate that the assays are very robust.

Fig. 7 shows examples of real time PCR curves for Molecular Beacons with a 9-mer and a 10-mer recognition site. Panel (A): Molecular beacon probe with a 10-mer recognition site detecting the 469 SSA4 amplicon. Signal was only obtained in the sample where SSA4 cDNA

was added (2 x 10^7 copies). A C_t value of 24.0 was obtained. A similar experiment with a molecular beacon having a 9-mer recognition site detecting the 570 SSA4 amplicon is shown in panel (B). Signal was only obtained when SSA4 cDNA was added (2 x 10^7 copies).

Fig. 8 shows an example of a real time PCR curve for a SYBR-probe with a 9-mer recognition site targeting the 570 SSA4 amplicon. Signal was only obtained in the sample where SSA4 cDNA was added (2 x 10^7 copies), whereas no signal was detected without addition of template.

Fig. 9 shows a calibration curve for three different 9-mer multi-probes using a dual labelled probe assay principle. Detection of different copy number levels of the SSA4 cDNA by the three dual labelled probes. The threshold cycle nr defines the cycle number at which signal was first detected for the respective PCR. Slope (a) and correlation coefficients (R^2) of the three linear regression lines are: a = -3.456 & R^2 = 0.9999 (Dual-labelled-469), a = -3.468 & R^2 = 0.9981 (Dual-labelled-570), and a = -3.499 & R^2 = 0.9993 (Dual-labelled-671).

Fig. 10 shows the use of 9-mer dual labelled multi-probes to quantify a heat shock protein before and after-exposure to heat shock in a wild type yeast strain as well as a mutant strain where the corresponding gene has been deleted. Real time detection of SSA4 transcript levels in wild type (wt) yeast and in the SSA4 knockout mutant with the Dual-labelled-570 probe is shown. The different strains were either cultured at 30°C till harvest (- HS) or they were exposed to 40°C for 30 minutes prior to harvest. The Dual-labelled-570 probe was used in this example. The transcript was only detected in the wt type strain, where it was most abundant in the + HS culture. C_t values were 26,1 and 30.3 for the + HS and the - HS culture, respectively.

Fig. 11 shows an example of how more than one gene can be detected by the same 9-mer probe while nucleic acid molecules without the probe target sequence (i.e. complementary to the recognition sequence) will not be detected. In (a) Dual-labelled-469 detects both the SSA4 (469 amplicon) and the POL5 transcript with C_t values of 29.7 and 30.1, respectively. No signal was detected from the APG9 and HSP82 transcripts. In (b) Dual-labelled-570 detects both the SSA4 (570 amplicon) and the APG9 transcript with C_t values of 31.3 and 29.2 respectively. No signal is detected from the POL5 and HSP82 transcripts. In (c) probe Dual-labelled-671 detected both the SSA4 (671 amplicon) and the HSP82 transcript with C_t values of 29.8 and 25.6 respectively. No signal was detected from the POL5 and APG9 transcripts. The amplicon produced in the different PCRs is indicated in the legend. The same amount of cDNA was used as in the experiments depicted in Figure 10. Only cDNA from non-heat shocked wild type yeast was used.

Fig. 12 shows agarose gel electrophoresis of a fraction of the amplicons generated in the PCR reactions shown in the example of Fig. 11, demonstrating that the probes are specific for target sequences comprising the recognition sequence but do not hybridize to nucleic acid molecules which do not comprise the target sequence. In lane 1 contain the SSA4-469 amplicon (81 bp), lane 2 contains the POL5 amplicon (94 bp), lane 3 contains the APG9 amplicon (97 bp) and lane 4 contains the HSP82 amplicon (88 bp). Lane M contains a 50 bp ladder as size indicator. It is clear that a product was formed in all four cases; however, only amplificates containing the correct multi-probe target sequence (i.e.SSA4-467 and POL5) were detected by the dual labelled probe 467. That two different amplificates were indeed produced and detected is evident from the size difference in the detected fragments from lane 1 and 2.

Fig. 13: Preferred target sequences.

Fig. 14: Further Preferred target sequences.

Fig. 15: Longmers (positive controls). The sequences are set forth in SEQ ID NOs. 32-46.

Fig. 16: Procedure for the selection of probes and the designing of primers for qPCR.

Fig. 17: Source code for the program used in the calculation of a multi-probe dataset.

DETAILED DESCRIPTION

The present invention relates to short oligonucleotide probes or multi-probes, chosen and designed to detect, classify or characterize, and/or quantify many different target nucleic acid molecules. These multi-probes comprise at least one non-natural modification (e.g. such as LNA nucleotide) for increasing the binding affinity of the probes for a recognition sequence, which is a subsequence of the target nucleic acid molecules. The target nucleic acid molecules are otherwise different outside of the recognition sequence.

In one aspect, the multi-probes comprise at least one nucleotide modified with a chemical moiety for increasing binding affinity of the probes for a recognition sequence, which is a subsequence of the target nucleic acid sequence. In another aspect, the probes comprise both at least one non-natural nucleotide and at least one nucleotide modified with a chemical moiety. In a further aspect, the at least one non-natural nucleotide is modified by the chemical moiety. The invention also provides kits, libraries and other compositions comprising the probes.

The invention further provides methods for choosing and designing suitable oligonucleotide probes for a given mixture of target sequences, ii) individual probes with these abilities, and iii) libraries of such probes chosen and designed to be able to detect, classify, and/or quantify the largest number of target nucleotides with the smallest number of probe sequences. Each probe according to the invention is thus able to bind many different targets, but may be used to create a specific assay when combined with a set of specific primers in PCR assays.

Preferred oligonucleotides of the invention are comprised of about 8 to 9 nucleotide units, a substantial portion of which comprises stabilizing nucleotides, such as LNA nucleotides. A preferred library contains approximately 100 of these probes chosen and designed to characterize a specific pool of nucleic acids, such as mRNA, cDNA or genomic DNA. Such a library may be used in a wide variety of applications, e.g., gene expression analyses, SNP detection, and the like. (See, e.g., Fig. 1).

Definitions

The following definitions are provided for specific terms, which are used in the disclosure of the present invention:

As used herein, the singular form "a", "an" and "the" include plural references unless the context clearly dictates otherwise. For example, the term "a cell" includes a plurality of cells, including mixtures thereof. The term "a nucleic acid molecule" includes a plurality of nucleic acid molecules.

As used herein, the term "transcriptome" refers to the complete collection of transcribed elements of the genome of any species.

In addition to mRNAs, it also represents non-coding RNAs which are used for structural and regulatory purposes.

As used herein, the term "amplicon refers to small, replicating DNA fragments.

As used herein, a "sample" refers to a sample of tissue or fluid isolated from an organism or organisms, including but not limited to, for example, skin, plasma, serum, spinal fluid, lymph fluid, synovial fluid, urine, tears, blood cells, organs, tumors, and also to samples of in vitro cell culture constituents (including but not limited to conditioned medium resulting from the growth of cells in cell culture medium, recombinant cells and cell components).

As used herein, an "organism" refers to a living entity, including but not limited to, for example, human, mouse, rat, *Drosophila*, *C. elegans*, yeast, Arabidopsis, zebra fish, primates, domestic animals, etc.

By the term "SBC nucleobases" is meant "Selective Binding Complementary" nucleobases, i.e. modified nucleobases that can make stable hydrogen bonds to their complementary nucleobases, but are unable to make stable hydrogen bonds to other SBC nucleobases. As an example, the SBC nucleobase A', can make a stable hydrogen bonded pair with its complementary unmodified nucleobase, T. Likewise, the SBC nucleobase T' can make a stable hydrogen bonded pair with its complementary unmodified nucleobase, A. However, the SBC nucleobases A' and T' will form an unstable hydrogen bonded pair as compared to the basepairs A'-T and A-T'. Likewise, a SBC nucleobase of C is designated C' and can make a stable hydrogen bonded pair with its complementary unmodified nucleobase G, and a SBC nucleobase of G is designated G' and can make a stable hydrogen bonded pair with its complementary unmodified nucleobase C, yet C' and G' will form an unstable hydrogen bonded pair as compared to the basepairs C'-G and C-G'. A stable hydrogen bonded pair is obtained when 2 or more hydrogen bonds are formed e.g. the pair between A' and T, A and T', C and G', and C' and G. An unstable hydrogen bonded pair is obtained when 1 or no hydrogen bonds is formed e.g. the pair between A' and T', and C' and G'.

Especially interesting SBC nucleobases are 2,6-diaminopurine (A', also called D) together with 2-thio-uracil (U', also called ^{2S}U)(2-thio-4-oxo-pyrimidine) and 2-thio-thymine (T', also called ^{2S}T)(2-thio-4-oxo-5-methyl-pyrimidine). Fig. 4 illustrates that the pairs A-^{2S}T and D-T have 2 or more than 2 hydrogen bonds whereas the D-^{2S}T pair forms a single (unstable) hydrogen bond. Likewise the SBC nucleobases pyrrolo-[2,3-d]pyrimidine-2(3H)-one (C', also called PyrroloPyr) and hypoxanthine (G', also called I)(6-oxo-purine) are shown in Fig. 9 where the pairs PyrroloPyr-G and C-I have 2 hydrogen bonds each whereas the PyrroloPyr-I pair forms a single hydrogen bond.

By "SBC LNA oligomer" is meant a "LNA oligomer" containing at least one "LNA unit" where the nucleobase is a "SBC nucleobase". By "LNA unit with an SBC nucleobase" is meant a "SBC LNA monomer". Generally speaking SBC LNA oligomers include oligomers that besides the SBC LNA monomer(s) contain other modified or naturally-occuring nucleotides or nucleosides. By "SBC monomer" is meant a non-LNA monomer with a SBC nucleobase. By "isosequential oligonucleotide" is meant an oligonucleotide with the same sequence in a Watson-Crick sense as the corresponding modified oligonucleotide e.g. the sequences agTtcATg is equal to agTscD^{2S}Ug where s is equal to the SBC DNA monomer 2-thio-t or 2-thio-u, D is equal to the SBC LNA monomer LNA-D and ^{2S}U is equal to the SBC LNA monomer LNA ^{2S}U.

As used herein, the terms "nucleic acid", "polynucleotide" and "oligonucleotide" refer to primers, probes, oligomer fragments to be detected, oligomer controls and unlabelled blocking oligomers and shall be generic to polydeoxyribonucleotides (containing 2-deoxy-D-ribose), to polyribonucleotides (containing D-ribose), and to any other type of polynucleotide which is an N glycoside of a purine or pyrimidine base, or modified purine or pyrimidine bases. There is no intended distinction in length between the term "nucleic acid", "polynucleotide" and "oligonucleotide", and these terms will be used interchangeably. These terms refer only to the primary structure of the molecule. Thus, these terms include double- and single-stranded DNA, as well as double- and single stranded RNA. The oligonucleotide is comprised of a sequence of approximately at least 3 nucleotides, preferably at least about 6 nucleotides, and more preferably at least about 8 - 30 nucleotides corresponding to a region of the designated nucleotide sequence. "Corresponding" means identical to or complementary to the designated sequence.

The oligonucleotide is not necessarily physically derived from any existing or natural sequence but may be generated in any manner, including chemical synthesis, DNA replication, reverse transcription or a combination thereof. The terms "oligonucleotide" or "nucleic acid" intend a polynucleotide of genomic DNA or RNA, cDNA, semi synthetic, or synthetic origin which, by virtue of its origin or manipulation: (1) is not associated with all or a portion of the polynucleotide with which it is associated in nature; and/or (2) is linked to a polynucleotide other than that to which it is linked in nature; and (3) is not found in nature.

Because mononucleotides are reacted to make oligonucleotides in a manner such that the 5'. phosphate of one mononucleotide pentose ring is attached to the 3' oxygen of its neighbour in one direction via a phosphodiester linkage, an end of an oligonucleotide is referred to as the "5' end" if its 5' phosphate is not linked to the 3' oxygen of a mononucleotide pentose ring and as the "3' end" if its 3' oxygen is not linked to a 5' phosphate of a subsequent mononucleotide pentose ring. As used herein, a nucleic acid sequence, even if internal to a larger oligonucleotide, also may be said to have a 5' and 3' ends.

When two different, non-overlapping oligonucleotides anneal to different regions of the same linear complementary nucleic acid sequence, the 3' end of one oligonucleotide points toward the 5' end of the other; the former may be called the "upstream" oligonucleotide and the latter the "downstream" oligonucleotide.

The term "primer" may refer to more than one primer and refers to an oligonucleotide, whether occurring naturally, as in a purified restriction digest, or produced synthetically, which is capable of acting as a point of initiation of synthesis along a complementary strand when placed under conditions in which synthesis of a primer extension product which is com-

plementary to a nucleic acid strand is catalyzed. Such conditions include the presence of four different deoxyribonucleoside triphosphates and a polymerization-inducing agent such as DNA polymerase or reverse transcriptase, in a suitable buffer ("buffer" includes substituents which are cofactors, or which affect pH, ionic strength, etc.), and at a suitable temperature. The primer is preferably single-stranded for maximum efficiency in amplification.

As used herein, the terms "PCR reaction", "PCR amplification", "PCR" and "real-time PCR" are interchangeable terms used to signify use of a nucleic acid amplification system, which multiplies the target nucleic acids being detected. Examples of such systems include the polymerase chain reaction (PCR) system and the ligase chain reaction (LCR) system. Other methods recently described and known to the person of skill in the art are the nucleic acid sequence based amplification (NASBA™, Cangene, Mississauga, Ontario) and Q Beta Replicase systems. The products formed by said amplification reaction may or may not be monitored in real time or only after the reaction as an end point measurement.

The complement of a nucleic acid sequence as used herein refers to an oligonucleotide which, when aligned with the nucleic acid sequence such that the 5' end of one sequence is paired with the 3' end of the other, is in "antiparallel association." Bases not commonly found in natural nucleic acids may be included in the nucleic acids of the present invention include, for example, inosine and 7-deazaguanine. Complementarity may not be perfect; stable duplexes may contain mismatched base pairs or unmatched bases. Those skilled in the art of nucleic acid technology can determine duplex stability empirically considering a number of variables including, for example, the length of the oligonucleotide, percent concentration of cytosine and guanine bases in the oligonucleotide, ionic strength, and incidence of mismatched base pairs.

Stability of a nucleic acid duplex is measured by the melting temperature, or "T_m". The T_m of a particular nucleic acid duplex under specified conditions is the temperature at which half of the base pairs have disassociated.

As used herein, the term "probe" refers to a labelled oligonucleotide, which forms a duplex structure with a sequence in the target nucleic acid, due to complementarity of at least one sequence in the probe with a sequence in the target region. The probe, preferably, does not contain a sequence complementary to sequence(s) used to prime the polymerase chain reaction. Generally the 3' terminus of the probe will be "blocked" to prohibit incorporation of the probe into a primer extension product. "Blocking" may be achieved by using non-complementary bases or by adding a chemical moiety such as biotin or even a phosphate group to the 3' hydroxyl of the last nucleotide, which may, depending upon the selected moiety, may serve a dual purpose by also acting as a label.

The term "label" as used herein refers to any atom or molecule which can be used to provide a detectable (preferably quantifiable) signal, and which can be attached to a nucleic acid or protein. Labels may provide signals detectable by fluorescence, radioactivity, colorimetric, X-ray diffraction or absorption, magnetism, enzymatic activity, and the like.

As defined herein, "5'→3' nuclease activity" or "5' to 3' nuclease activity" refers to that activity of a template-specific nucleic acid polymerase including either a 5'→3' exonuclease activity traditionally associated with some DNA polymerases whereby nucleotides are removed from the 5' end of an oligonucleotide in a sequential manner, (i.e., E. coli DNA polymerase I has this activity whereas the Klenow fragment does not), or a 5'→3' endonuclease activity wherein cleavage occurs more than one nucleotide from the 5' end, or both.

As used herein, the term "thermo stable nucleic acid polymerase" refers to an enzyme which is relatively stable to heat when compared, for example, to nucleotide polymerases from E. coli and which catalyzes the polymerization of nucleosides. Generally, the enzyme will initiate synthesis at the 3'-end of the primer annealed to the target sequence, and will proceed in the 5'-direction along the template, and if possessing a 5' to 3' nuclease activity, hydrolyzing or displacing intervening, annealed probe to release both labelled and unlabelled probe fragments or intact probe, until synthesis terminates. A representative thermo stable enzyme isolated from Thermus aquaticus (Tag) is described in U.S. Pat. No. 4,889,818 and a method for using it in conventional PCR is described in Saiki et al., (1988), Science 239:487.

The term "nucleobase" covers the naturally occurring nucleobases adenine (A), guanine (G), cytosine (C), thymine (T) and uracil (U) as well as non-naturally occurring nucleobases such as xanthine, diaminopurine, 8-oxo-N^6-methyladenine, 7-deazaxanthine, 7-deazaguanine, N^4,N^4-ethanocytosin, N^6,N^6-ethano-2,6-diaminopurine, 5-methylcytosine, 5-(C^3-C^6)-alkynyl-cytosine, 5-fluorouracil, 5-bromouracil, pseudoisocytosine, 2-hydroxy-5-methyl-4-triazolopyridin, isocytosine, isoguanine, inosine and the "non-naturally occurring" nucleobases described in Benner et al., U.S. Patent No. 5,432,272 and Susan M. Freier and Karl-Heinz Altmann, *Nucleic Acid Research,25*: 4429-4443, 1997. The term "nucleobase" thus includes not only the known purine and pyrimidine heterocycles, but also heterocyclic analogues and tautomers thereof. Further naturally and non naturally occurring nucleobases include those disclosed in U.S. Patent No. 3,687,808; in chapter 15 by Sanghvi, in *Antisense Research and Application*, Ed. S. T. Crooke and B. Lebleu, CRC Press, 1993; in Englisch, et al., *Angewandte Chemie, International Edition*, 30: 613-722, 1991 (see, especially pages 622 and 623, and in the *Concise Encyclopedia of Polymer Science and Engineering*, J. I. Kroschwitz Ed., John Wiley & Sons, pages 858-859, 1990, Cook, *Anti-Cancer DrugDesign* 6: 585-607, 1991, each of which are hereby incorporated by reference in their entirety).

The term "nucleosidic base" or "nucleobase analogue" is further intended to include heterocyclic compounds that can serve as like nucleosidic bases including certain "universal bases" that are not nucleosidic bases in the most classical sense but serve as nucleosidic bases. Especially mentioned as a universal base is 3-nitropyrrole a 5-nitroindole. Other preferred compounds include pyrene and pyridyloxazole derivatives, pyrenyl, pyrenylmethylglycerol derivatives and the like. Other preferred universal bases include, pyrrole, diazole or triazole derivatives, including those universal bases known in the art.

By "universal base" is meant a naturally-occurring or desirably a non-naturally occurring compound or moiety that can pair with a natural base (e.g., adenine, guanine, cytosine, uracil, and/or thymine), and that has a T_m differential of 15, 12, 10, 8, 6, 4, or 2°C or less as described herein.

By "oligonucleotide," "oligomer," or "oligo" is meant a successive chain of monomers (e.g., glycosides of heterocyclic bases) connected via internucleoside linkages. The linkage between two successive monomers in the oligo consist of 2 to 4, desirably 3, groups/atoms selected from $-CH_2-$, $-O-$, $-S-$, $-NR^H-$, $>C=O$, $>C=NR^H$, $>C=S$, $-Si(R'')_2-$, $-SO-$, $-S(O)_2-$, $-P(O)_2-$, $-PO(BH_3)-$, $-P(O,S)-$, $-P(S)_2-$, $-PO(R'')-$, $-PO(OCH_3)-$, and $-PO(NHR^H)-$, where R^H is selected from hydrogen and C_{1-4}-alkyl, and R'' is selected from C_{1-6}-alkyl and phenyl. Illustrative examples of such linkages are $-CH_2-CH_2-CH_2-$, $-CH_2-CO-CH_2-$, $-CH_2-CHOH-CH_2-$, $-O-CH_2-O-$, $-O-CH_2-CH_2-$, $-O-CH_2-CH=$ (including R^5 when used as a linkage to a succeeding monomer), $-CH_2-CH_2-O-$, $-NR^H-CH_2-CH_2-$, $-CH_2-CH_2-NR^H-$, $-CH_2-NR^H-CH_2-$, $-O-CH_2-CH_2-NR^H-$, $-NR^H-CO-O-$, $-NR^H-CO-NR^H-$, $-NR^H-CS-NR^H-$, $-NR^H-C(=NR^H)-NR^H-$, $-NR^H-CO-CH_2-NR^H-$, $-O-CO-O-$, $-O-CO-CH_2-O-$, $-O-CH_2-CO-O-$, $-CH_2-CO-NR^H-$, $-O-CO-NR^H-$, $-NR^H-CO-CH_2-$, $-O-CH_2-CO-NR^H-$, $-O-CH_2-CH_2-NR^H-$, $-CH=N-O-$, $-CH_2-NR^H-O-$, $-CH_2-O-N=$ (including R^5 when used as a linkage to a succeeding monomer), $-CH_2-O-NR^H-$, $-CO-NR^H-CH_2-$, $-CH_2-NR^H-O-$, $-CH_2-NR^H-CO-$, $-O-NR^H-CH_2-$, $-O-NR^H-$, $-O-CH_2-S-$, $-S-CH_2-O-$, $-CH_2-CH_2-S-$, $-O-CH_2-CH_2-S-$, $-S-CH_2-CH=$ (including R^5 when used as a linkage to a succeeding monomer), $-S-CH_2-CH_2-$, $-S-CH_2-CH_2-O-$, $-S-CH_2-CH_2-S-$, $-CH_2-S-CH_2-$, $-CH_2-SO-CH_2-$, $-CH_2-SO_2-CH_2-$, $-O-SO-O-$, $-O-S(O)_2-O-$, $-O-S(O)_2-CH_2-$, $-O-S(O)_2-NR^H-$, $-NR^H-S(O)_2-CH_2-$, $-O-S(O)_2-CH_2-$, $-O-P(O)_2-O-$, $-O-P(O,S)-O-$, $-O-P(S)_2-O-$, $-S-P(O)_2-O-$, $-S-P(O,S)-O-$, $-S-P(S)_2-O-$, $-O-P(O)_2-S-$, $-O-P(O,S)-S-$, $-O-P(S)_2-S-$, $-S-P(O)_2-S-$, $-S-P(O,S)-S-$, $-S-P(S)_2-S-$, $-O-PO(R'')-O-$, $-O-PO(OCH_3)-O-$, $-O-PO(OCH_2CH_3)-O-$, $-O-PO(OCH_2CH_2S-R)-O-$, $-O-PO(BH_3)-O-$, $-O-PO(NHR^N)-O-$, $-O-P(O)_2-NR^H-$, $-NR^H-P(O)_2-O-$, $-O-P(O,NR^H)-O-$, $-CH_2-P(O)_2-O-$, $-O-P(O)_2-CH_2-$, and $-O-Si(R'')_2-O-$; among which $-CH_2-CO-NR^H-$, $-CH_2-NR^H-O-$, $-S-CH_2-O-$, $-O-P(O)_2-O-$, $-O-P(O,S)-O-$, $-O-P(S)_2-O-$, $-NR^H-P(O)_2-O-$, $-O-P(O,NR^H)-O-$, $-O-PO(R'')-O-$, $-O-PO(CH_3)-O-$, and $-O-PO(NHR^N)-O-$, where R^H is selected form hydrogen and C_{1-4}-alkyl, and R'' is selected from C_{1-6}-alkyl and phenyl, are especially desirable. Further illustrative examples are given in Mesmaeker *et. al.*, Current Opinion in Structural Biology 1995, 5, 343-355 and Susan M. Freier and Karl-Heinz Altmann, Nucleic Acids

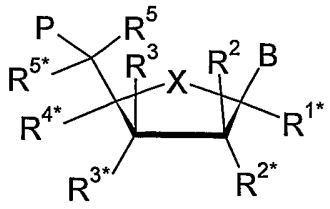
Research, 1997, vol 25, pp 4429-4443. The left-hand side of the internucleoside linkage is bound to the 5-membered ring as substituent P^* at the 3'-position, whereas the right-hand side is bound to the 5'-position of a preceding monomer.

By "LNA unit" is meant an individual LNA monomer (e.g., an LNA nucleoside or LNA nucleotide) or an oligomer (e.g., an oligonucleotide or nucleic acid) that includes at least one LNA monomer. LNA units as disclosed in WO 99/14226 are in general particularly desirable modified nucleic acids for incorporation into an oligonucleotide of the invention. Additionally, the nucleic acids may be modified at either the 3' and/or 5' end by any type of modification known in the art. For example, either or both ends may be capped with a protecting group, attached to a flexible linking group, attached to a reactive group to aid in attachment to the substrate surface, etc. Desirable LNA units and their method of synthesis also are disclosed in WO 00/47599, US 6,043,060, US 6,268,490, PCT/JP98/00945, WO 0107455, WO 0100641, WO 9839352, WO 0056746, WO 0056748, WO 0066604, Morita *et al.*, Bioorg. Med. Chem. Lett. 12(1):73-76, 2002; Hakansson *et al.*, Bioorg. Med. Chem. Lett. 11(7):935-938, 2001; Koshkin *et al.*, J. Org. Chem. 66(25):8504-8512, 2001; Kvaerno *et al.*, J. Org. Chem. 66(16):5498-5503, 2001; Hakansson *et al.*, J. Org. Chem. 65(17):5161-5166, 2000; Kvaerno *et al.*, J. Org. Chem. 65(17):5167-5176, 2000; Pfundheller *et al.*, Nucleosides Nucleotides 18(9):2017-2030, 1999; and Kumar *et al.*, Bioorg. Med. Chem. Lett. 8(16):2219-2222, 1998.

Preferred LNA monomers, also referred to as "oxy-LNA" are LNA monomers which include bicyclic compounds as disclosed in PCT Publication WO 03/020739 wherein the bridge between $R^{4'}$ and $R^{2'}$ as shown in formula (I) below together designate $-CH_2-O-$ (methyloxy LNA) or $-CH_2-CH_2-O-$ (ethyloxy LNA, also designated ENA).

Further preferred LNA monomers are designated "thio-LNA" or "amino-LNA" including bicyclic structures as disclosed in WO 99/14226, wherein the heteroatom in the bridge between $R^{4'}$ and $R^{2'}$ as shown in formula (I) below together designate $-CH_2-S-$, $-CH_2-CH_2-S-$, $-CH_2-NH-$ or $-CH_2-CH_2-NH-$.

By "LNA modified oligonucleotide" is meant a oligonucleotide comprising at least one LNA monomeric unit of formula (I), described *infra,* having the below described illustrative examples of modifications:



(I)

wherein X is selected from -O-, -S-, -N(R^N)-, -C(R^6R^{6*})-, -O-C(R^7R^{7*})-, -C(R^6R^{6*})-O-, -S-C(R^7R^{7*})-, -C(R^6R^{6*})-S-, -N(R^{N*})-C(R^7R^{7*})-, -C(R^6R^{6*})-N(R^{N*})-, and -C(R^6R^{6*})-C(R^7R^{7*}).

B is selected from a modified base as discussed above e.g. an optionally substituted carbocyclic aryl such as optionally substituted pyrene or optionally substituted pyrenylmethylglycerol, or an optionally substituted heteroalicylic or optionally substituted heteroaromatic such as optionally substituted pyridyloxazole, optionally substituted pyrrole, optionally substituted diazole or optionally substituted triazole moieties; hydrogen, hydroxy, optionally substituted C_{1-4}-alkoxy, optionally substituted C_{1-4}-alkyl, optionally substituted C_{1-4}-acyloxy, nucleobases, DNA intercalators, photochemically active groups, thermochemically active groups, chelating groups, reporter groups, and ligands.

P designates the radical position for an internucleoside linkage to a succeeding monomer, or a 5'-terminal group, such internucleoside linkage or 5'-terminal group optionally including the substituent R^5. One of the substituents R^2, R^{2*}, R^3, and R^{3*} is a group P* which designates an internucleoside linkage to a preceding monomer, or a 2'/3'-terminal group. The substituents of R^{1*}, R^{4*}, R^5, R^{5*}, R^6, R^{6*}, R^7, R^{7*}, R^N, and the ones of R^2, R^{2*}, R^3, and R^{3*} not designating P* each designates a biradical comprising about 1-8 groups/atoms selected from -C(R^aR^b)-, -C(R^a)=C(R^a)-, -C(R^a)=N-, -C(R^a)-O-, -O-, -Si(R^a)$_2$-, -C(R^a)-S, -S-, -SO_2-, -C(R^a)-N(R^b)-, -N(R^a)-, and >C=Q, wherein Q is selected from -O-, -S-, and -N(R^a)-, and R^a and R^b each is independently selected from hydrogen, optionally substituted C_{1-12}-alkyl, optionally substituted C_{2-12}-alkenyl, optionally substituted C_{2-12}-alkynyl, hydroxy, C_{1-12}-alkoxy, C_{2-12}-alkenyloxy, carboxy, C_{1-12}-alkoxycarbonyl, C_{1-12}-alkylcarbonyl, formyl, aryl, aryloxy-carbonyl, aryloxy, arylcarbonyl, heteroaryl, hetero-aryloxy-carbonyl, heteroaryloxy, heteroarylcarbonyl, amino, mono- and di(C_{1-6}-alkyl)amino, carbamoyl, mono- and di(C_{1-6}-alkyl)-amino-carbonyl, amino-C_{1-6}-alkyl-aminocarbonyl, mono- and di(C_{1-6}-alkyl)amino-C_{1-6}-alkyl-aminocarbonyl, C_{1-6}-alkyl-carbonylamino, carbamido, C_{1-6}-alkanoyloxy, sulphono, C_{1-6}-alkylsulphonyloxy, nitro, azido, sulphanyl, C_{1-6}-alkylthio, halogen, DNA intercalators, photochemically active groups, thermochemically active groups, chelating groups, reporter groups, and ligands, where aryl and heteroaryl may be optionally substituted, and where two geminal substituents R^a and R^b together may designate optionally substituted methylene (=CH_2), and wherein two non-geminal or geminal substituents selected from R^a, R^b, and any of the substituents R^{1*}, R^2, R^{2*}, R^3, R^{3*}, R^{4*}, R^5, R^{5*}, R^6 and R^{6*}, R^7, and R^{7*} which are present and not involved in P,

P^* or the biradical(s) together may form an associated biradical selected from biradicals of the same kind as defined before; the pair(s) of non-geminal substituents thereby forming a mono- or bicyclic entity together with (i) the atoms to which said non-geminal substituents are bound and (ii) any intervening atoms.

Each of the substituents R^{1*}, R^2, R^{2*}, R^3, R^{4*}, R^5, R^{5*}, R^6 and R^{6*}, R^7, and R^{7*} which are present and not involved in P, P^* or the biradical(s), is independently selected from hydrogen, optionally substituted C_{1-12}-alkyl, optionally substituted C_{2-12}-alkenyl, optionally substituted C_{2-12}-alkynyl, hydroxy, C_{1-12}-alkoxy, C_{2-12}-alkenyloxy, carboxy, C_{1-12}-alkoxycarbonyl, C_{1-12}-alkylcarbonyl, formyl, aryl, aryloxy-carbonyl, aryloxy, arylcarbonyl, heteroaryl, heteroaryl-oxy-carbonyl, heteroaryloxy, heteroarylcarbonyl, amino, mono- and di(C_{1-6}-alkyl)amino, carbamoyl, mono- and di(C_{1-6}-alkyl)-amino-carbonyl, amino-C_{1-6}-alkyl-aminocarbonyl, mono- and di(C_{1-6}-alkyl)amino-C_{1-6}-alkyl-aminocarbonyl, C_{1-6}-alkyl-carbonylamino, carbamido, C_{1-6}-alkanoyloxy, sulphono, C_{1-6}-alkylsulphonyloxy, nitro, azido, sulphanyl, C_{1-6}-alkylthio, halogen, DNA intercalators, photochemically active groups, thermochemically active groups, chelating groups, reporter groups, and ligands, where aryl and heteroaryl may be optionally substituted, and where two geminal substituents together may designate oxo, thioxo, imino, or optionally substituted methylene, or together may form a spiro biradical consisting of a 1-5 carbon atom(s) alkylene chain which is optionally interrupted and/or terminated by one or more heteroatoms/groups selected from -O-, -S-, and -(NR^N)- where R^N is selected from hydrogen and C_{1-4}-alkyl, and where two adjacent (non-geminal) substituents may designate an additional bond resulting in a double bond; and R^{N*}, when present and not involved in a biradical, is selected from hydrogen and C_{1-4}-alkyl; and basic salts and acid addition salts thereof.

Exemplary 5', 3', and/or 2' terminal groups include -H, -OH, halo (*e.g.*, chloro, fluoro, iodo, or bromo), optionally substituted aryl, (*e.g.*, phenyl or benzyl), alkyl (*e.g.*, methyl or ethyl), alkoxy (*e.g.*, methoxy), acyl (*e.g.* acetyl or benzoyl), aroyl, aralkyl, hydroxy, hydroxyalkyl, alkoxy, aryloxy, aralkoxy, nitro, cyano, carboxy, alkoxycarbonyl, aryloxycarbonyl, aralkoxy-carbonyl, acylamino, aroylamino, alkylsulfonyl, arylsulfonyl, heteroarylsulfonyl, alkylsulfinyl, arylsulfinyl, heteroarylsulfinyl, alkylthio, arylthio, heteroarylthio, aralkylthio, heteroaralkylthio, amidino, amino, carbamoyl, sulfamoyl, alkene, alkyne, protecting groups (e.g., silyl, 4,4'-dimethoxytrityl, monomethoxytrityl, or trityl(triphenylmethyl)), linkers (e.g., a linker containing an amine, ethylene glycol, quinone such as anthraquinone), detectable labels (e.g., radiolabels or fluorescent labels), and biotin.

It is understood that references herein to a nucleic acid unit, nucleic acid residue, LNA unit, or similar term are inclusive of both individual nucleoside units and nucleotide units and nucleoside units and nucleotide units within an oligonucleotide.

A "modified base" or other similar term refers to a composition (e.g., a non-naturally occurring nucleobase or nucleosidic base), which can pair with a natural base (e.g., adenine, guanine, cytosine, uracil, and/or thymine) and/or can pair with a non-naturally occurring nucleobase or nucleosidic base. Desirably, the modified base provides a T_m differential of 15, 12, 10, 8, 6, 4, or 2°C or less as described herein. Exemplary modified bases are described in EP 1 072 679 and WO 97/12896.

The term "chemical moiety" refers to a part of a molecule. "Modified by a chemical moiety" thus refer to a modification of the standard molecular structure by inclusion of an unusual chemical structure. The attachment of said structure can be covalent or non-covalent.

The term "inclusion of a chemical moiety" in an oligonucleotide probe thus refers to attachment of a molecular structure. Such as chemical moiety include but are not limited to covalently and/or non-covalently bound minor groove binders (MGB) and/or intercalating nucleic acids (INA) selected from a group consisting of asymmetric cyanine dyes, DAPI, SYBR Green I, SYBR Green II, SYBR Gold, PicoGreen, thiazole orange, Hoechst 33342, Ethidium Bromide, 1-O-(1-pyrenylmethyl)glycerol and Hoechst 33258. Other chemical moieties include the modified nucleobases, nucleosidic bases or LNA modified oligonucleotides.

The term "Dual labelled probe" refers to an oligonucleotide with two attached labels. In one aspect, one label is attached to the 5′ end of the probe molecule, whereas the other label is attached to the 3′ end of the molecule. A particular aspect of the invention contain a fluorescent molecule attached to one end and a molecule which is able to quench this fluorophore by Fluorescence Resonance Energy Transfer (FRET) attached to the other end. 5′ nuclease assay probes and some Molecular Beacons are examples of Dual labelled probes.

The term "5′ nuclease assay probe" refers to a dual labelled probe which may be hydrolyzed by the 5′-3′ exonuclease activity of a DNA polymerase. A 5′ nuclease assay probes is not necessarily hydrolyzed by the 5′-3′ exonuclease activity of a DNA polymerase under the conditions employed in the particular PCR assay. The name "5′ nuclease assay" is used regardless of the degree of hydrolysis observed and does not indicate any expectation on behalf of the experimenter. The term "5′ nuclease assay probe" and "5′ nuclease assay" merely refers to assays where no particular care has been taken to avoid hydrolysis of the involved probe. "5′ nuclease assay probes" are often referred to as a "TaqMan assay probes", and the "5′ nuclease assay" as "TaqMan assay". These names are used interchangeably in this application.

The term "oligonucleotide analogue" refers to a nucleic acid binding molecule capable of recognizing a particular target nucleotide sequence. A particular oligonucleotide analogue is peptide nucleic acid (PNA) in which the sugar phosphate backbone of an oligonucleotide is

replaced by a protein like backbone. In PNA, nucleobases are attached to the uncharged polyamide backbone yielding a chimeric pseudopeptide-nucleic acid structure, which is homomorphous to nucleic acid forms.

The term "Molecular Beacon" refers to a single or dual labelled probe which is not likely to be affected by the 5'-3' exonuclease activity of a DNA polymerase. Special modifications to the probe, polymerase or assay conditions have been made to avoid separation of the labels or constituent nucleotides by the 5'-3' exonuclease activity of a DNA polymerase. The detection principle thus rely on a detectable difference in label elicited signal upon binding of the molecular beacon to its target sequence. In one aspect of the invention the oligonucleotide probe forms an intramolecular hairpin structure at the chosen assay temperature mediated by complementary sequences at the 5'- and the 3'-end of the oligonucleotide. The oligonucleotide may have a fluorescent molecule attached to one end and a molecule attached to the other, which is able to quench the fluorophore when brought into close proximity of each other in the hairpin structure. In another aspect of the invention, a hairpin structure is not formed based on complementary structure at the ends of the probe sequence instead the detected signal change upon binding may result from interaction between one or both of the labels with the formed duplex structure or from a general change of spatial conformation of the probe upon binding – or from a reduced interaction between the labels after binding. A particular aspect of the molecular beacon contain a number of LNA residues to inhibit hydrolysis by the 5'-3' exonuclease activity of a DNA polymerase.

The term "multi-probe" as used herein refers to a probe which comprises a recognition segment which is a probe sequence sufficiently complementary to a recognition sequence in a target nucleic acid molecule to bind to the sequence under moderately stringent conditions and/or under conditions suitable for PCR, 5' nuclease assay and/or Molecular Beacon analysis (or generally any FRET-based method). Such conditions are well known to those of skill in the art. Preferably, the recognition sequence is found in a plurality of sequences being evaluated, e.g., such as a transcriptome. A multi-probe according to the invention may comprise a non-natural nucleotide ("a stabilizing nucleotide") and may have a higher binding affinity for the recognition sequence than a probe comprising an identical sequence but without the stabilizing modification. Preferably, at least one nucleotide of a multi-probe is modified by a chemical moiety (e.g., covalently or otherwise stably associated with during at least hybridization stages of a PCR reaction) for increasing the binding affinity of the recognition segment for the recognition sequence.

As used herein, a multi-probe with an increased "binding affinity" for a recognition sequence than a probe which comprises the same sequence but which does not comprise a stabilizing nucleotide, refers to a probe for which the association constant (K_a) of the probe recognition

segment is higher than the association constant of the complementary strands of a double-stranded molecule. In another preferred embodiment, the association constant of the probe recognition segment is higher than the dissociation constant (K_d) of the complementary strand of the recognition sequence in the target sequence in a double stranded molecule.

A "multi-probe library" or "library of multi-probes" comprises a plurality of multi- probes, such that the sum of the probes in the library are able to recognise a major proportion of a transcriptome, including the most abundant sequences, such that about 60%, about 70%, about 80%, about 85%, more preferably about 90%, and still more preferably 95%, of the target nucleic acids in the transcriptome, are detected by the probes.

Monomers are referred to as being "complementary" if they contain nucleobases that can form hydrogen bonds according to Watson-Crick base-pairing rules (e.g. G with C, A with T or A with U) or other hydrogen bonding motifs such as for example diaminopurine with T, inosine with C, pseudoisocytosine with G, etc.

The term "succeeding monomer" relates to the neighboring monomer in the 5'-terminal direction and the "preceding monomer" relates to the neighboring monomer in the 3'-terminal direction.

As used herein, the term "target population" refers to a plurality of different sequences of nucleic acids, for example the genome or other nucleic acids from a particular species including the transcriptome of the genome, wherein the transcriptome refers to the complete collection of transcribed elements of the genome of any species. Normally, the number of different target sequences in a nucleic acid population is at least 100, but as will be clear the number is often much higher (more than 200, 500, 1000, and 10000 – in the case where the target population is a eukaryotic tran

As used herein, the term "target nucleic acid" refers to any relevant nucleic acid of a single specific sequence, e. g., a biological nucleic acid, e. g., derived from a patient, an animal (a human or non-human animal), a plant, a bacteria, a fungi, an archae, a cell, a tissue, an organism, etc. For example, where the target nucleic acid is derived from a bacteria, archae, plant, non-human animal, cell, fungi, or non-human organism, the method optionally further comprises selecting the bacteria, archae, plant, non-human animal, cell, fungi, or non-human organism based upon detection of the target nucleic acid. In one embodiment, the target nucleic acid is derived from a patient, e. g., a human patient. In this embodiment, the invention optionally further includes selecting a treatment, diagnosing a disease, or diagnosing a genetic predisposition to a disease, based upon detection of the target nucleic acid.

As used herein, the term "target sequence" refers to a specific nucleic acid sequence within any target nucleic acid.

The term "stringent conditions", as used herein, is the "stringency" which occurs within a range from about T_m-5°C (5°C below the melting temperature (T_m) of the probe) to about 20°C to 25°C below T_m. As will be understood by those skilled in the art, the stringency of hybridization may be altered in order to identify or detect identical or related polynucleotide sequences. Hybridization techniques are generally described in *Nucleic Acid Hybridization, A Practical Approach*, Ed. Hames, B. D. and Higgins, S. J., IRL Press, 1985; Gall and Pardue, *Proc. Natl. Acad. Sci., USA 63*: 378-383, 1969; and John, et al. *Nature* 223: 582-587, 1969.

Multi-probes

Referring now to Fig. 1B, a multi-probe according to the invention is preferably a short sequence probe which binds to a recognition sequence found in a plurality of different target nucleic acids, such that the multi-probe specifically hybridizes to the target nucleic acid but do not hybridize to any detectable level to nucleic acid molecules which do not comprise the recognition sequence. Preferably, a collection of multi-probes, or multi-probe library, is able to recognize a major proportion of a transcriptome, including the most abundant sequences, such as about 60%, about 70%, about 80%, about 85%, more preferably about 90%, and still more preferably 95%, of the target nucleic acids in the transcriptome, are detected by the probes. A multi-probe according to the invention comprises a "stabilizing modification" e.g. such as a non-natural nucleotide ("a stabilizing nucleotide") and has higher binding affinity for the recognition sequence than a probe comprising an identical sequence but without the stabilizing sequence. Preferably, at least one nucleotide of a multi-probe is modified by a chemical moiety (e.g., covalently or otherwise stably associated with the probe during at least hybridization stages of a PCR reaction) for increasing the binding affinity of the recognition segment for the recognition sequence.

In one aspect, a multi-probe of from 6 to 12 nucleotides comprises from 1 to 6 or even up to 12 stabilizing nucleotides, such as LNA nucleotides. An LNA enhanced probe library contains short probes that recognize a short recognition sequence (e.g., 8-9 nucleotides). LNA nucleobases can comprise α-LNA molecules (see, e.g., WO 00/66604) or xylo-LNA molecules (see, e.g., WO 00/56748).

In one aspect, it is preferred that the T_m of the multi-probe when bound to its recognition sequence is between about 55°C to about 70°C.

In another aspect, the multi-probes comprise one or more modified nucleobases. Modified base units may comprise a cyclic unit (e.g. a carbocyclic unit such as pyrenyl) that is joined to a nucleic unit, such as a 1'-position of furasonyl ring through a linker, such as a straight of branched chain alkylene or alkenylene group. Alkylene groups suitably having from 1 (i.e., -CH_2-) to about 12 carbon atoms, more typically 1 to about 8 carbon atoms, still more typically 1 to about 6 carbon atoms. Alkenylene groups suitably have one, two or three carbon-carbon double bounds and from 2 to about 12 carbon atoms, more typically 2 to about 8 carbon atoms, still more typically 2 to about 6 carbon atoms.

Multi-probes according to the invention are ideal for performing such assays as real-time PCR as the probes according to the invention are preferably less than about 25 nucleotides, less than about 15 nucleotides, less than about 10 nucleotides, e.g., 8 or 9 nucleotides. Preferably, a multi-probe can specifically hybridize with a recognition sequence within a target sequence under PCR conditions and preferably the recognition sequence is found in at least about 50, at least about 100, at least about 200, at least about 500 different target nucleic acid molecules. A library of multi-probes according to the invention will comprise multi-probes, which comprise non-identical recognition sequences, such that any two multi-probes hybridize to different sets of target nucleic acid molecules. In one aspect, the sets of target nucleic acid molecules comprise some identical target nucleic acid molecules, i.e., a target nucleic acid molecule comprising a gene sequence of interest may be bound by more than one multi-probe. Such a target nucleic acid molecule will contain at least two different recognition sequences which may overlap by one or more, but less than x nucleotides of a recognition sequence comprising x nucleotides.

In one aspect, a multi-probe library comprises a plurality of different multi-probes, each different probe localized at a discrete location on a solid substrate. As used herein, "localize" refers to being limited or addressed at the location such that hybridization event detected at the location can be traced to a probe of known sequence identity. A localized probe may or may not be stably associated with the substrate. For example, the probe could be in solution in the well of a microtiter plate and thus localized or addressed to the well. Alternatively, or additionally, the probe could be stably associated with the substrate such that it remains at a defined location on the substrate after one or more washes of the substrate with a buffer. For example, the probe may be chemically associated with the substrate, either directly or through a linker molecule, which may be a nucleic acid sequence, a peptide or other type of molecule, which has an affinity for molecules on the substrate.

Alternatively, the target nucleic acid molecules may be localized on a substrate (e.g., as a cell or cell lysate or nucleic acids dotted onto the substrate).

Once the appropriate sequences are determined, multi-LNA probes are preferably chemically synthesized using commercially available methods and equipment as described in the art (*Tetrahedron* 54: 3607-30, 1998). For example, the solid phase phosphoramidite method can be used to produce short LNA probes (Caruthers, et al., *Cold Spring Harbor Symp. Quant. Biol. 47*:411-418, 1982, Adams, et al., *J. Am. Chem. Soc. 105*: 661 (1983).

The determination of the extent of hybridization of multi-probes from a multi-probe library to one or more target sequences (preferably to a plurality of target sequences) may be carried out by any of the methods well known in the art. If there is no detectable hybridization, the extent of hybridization is thus 0. Typically, labelled signal nucleic acids are used to detect hybridization. Complementary nucleic acids or signal nucleic acids may be labelled by any one of several methods typically used to detect the presence of hybridized polynucleotides. The most common method of detection is the use of ligands, which bind to labelled antibodies, fluorophores or chemiluminescent agents. Other labels include antibodies, which can serve as specific binding pair members for a labelled ligand. The choice of label depends on sensitivity required, ease of conjugation with the probe, stability requirements, and available instrumentation.

LNA-containing-probes are typically labelled during synthesis. The flexibility of the phosphoramidite synthesis approach furthermore facilitates the easy production of LNAs carrying all commercially available linkers, fluorophores and labelling-molecules available for this standard chemistry. LNA may also be labelled by enzymatic reactions e.g. by kinasing.

Multi-probes according to the invention can comprise single labels or a plurality of labels. In one aspect, the plurality of labels comprise a pair of labels which interact with each other either to produce a signal or to produce a change in a signal when hybridization of the multi-probe to a target sequence occurs.

In another aspect, the multi-probe comprises a fluorophore moiety and a quencher moiety, positioned in such a way that the hybridized state of the probe can be distinguished from the unhybridized state of the probe by an increase in the fluorescent signal from the nucleotide. In one aspect, the multi-probe comprises, in addition to the recognition element, first and second complementary sequences, which specifically hybridize to each other, when the probe is not hybridized to a recognition sequence in a target molecule, bringing the quencher molecule in sufficient proximity to said reporter molecule to quench fluorescence of the reporter molecule. Hybridization of the target molecule distances the quencher from the reporter molecule and results in a signal, which is proportional to the amount of hybridization.

In another aspect, where polymerization of strands of nucleic acids can be detected using a polymerase with 5′ nuclease activity. Fluorophore and quencher molecules are incorporated into the probe in sufficient proximity such that the quencher quenches the signal of the fluorophore molecule when the probe is hybridized to its recognition sequence. Cleavage of the probe by the polymerase with 5′ nuclease activity results in separation of the quencher and fluorophore molecule, and the presence in increasing amounts of signal as nucleic acid sequences

In the present context, the term "label" means a reporter group, which is detectable either by itself or as a part of a detection series. Examples of functional parts of reporter groups are biotin, digoxigenin, fluorescent groups (groups which are able to absorb electromagnetic radiation, *e.g.* light or X-rays, of a certain wavelength, and which subsequently reemits the energy absorbed as radiation of longer wavelength; illustrative examples are DANSYL (5-dimethylamino)-1-naphthalenesulfonyl), DOXYL (N-oxyl-4,4-dimethyloxazolidine), PROXYL (N-oxyl-2,2,5,5-tetramethylpyrrolidine), TEMPO (N-oxyl-2,2,6,6-tetramethylpiperidine), dinitrophenyl, acridines, coumarins, Cy3 and Cy5 (trademarks for Biological Detection Systems, Inc.), erythrosine, coumaric acid, umbelliferone, Texas red, rhodamine, tetramethyl rhodamine, Rox, 7-nitrobenzo-2-oxa-1-diazole (NBD), pyrene, fluorescein, Europium, Ruthenium, Samarium, and other rare earth metals), radio isotopic labels, chemiluminescence labels (labels that are detectable via the emission of light during a chemical reaction), spin labels (a free radical (*e.g.* substituted organic nitroxides) or other paramagnetic probes (*e.g.* Cu^{2+}, Mg^{2+}) bound to a biological molecule being detectable by the use of electron spin resonance spectroscopy). Especially interesting examples are biotin, fluorescein, Texas Red, rhodamine, dinitrophenyl, digoxigenin, Ruthenium, Europium, Cy5, Cy3, etc.

Suitable samples of target nucleic acid molecule may comprise a wide range of eukaryotic and prokaryotic cells, including protoplasts; or other biological materials, which may harbour target nucleic acids. The methods are thus applicable to tissue culture animal cells, animal cells (e.g., blood, serum, plasma, reticulocytes, lymphocytes, urine, bone marrow tissue, cerebrospinal fluid or any product prepared from blood or lymph) or any type of tissue biopsy (e.g. a muscle biopsy, a liver biopsy, a kidney biopsy, a bladder biopsy, a bone biopsy, a cartilage biopsy, a skin biopsy, a pancreas biopsy, a biopsy of the intestinal tract, a thymus biopsy, a mammae biopsy, a uterus biopsy, a testicular biopsy, an eye biopsy or a brain biopsy, e.g., homogenized in lysis buffer), archival tissue nucleic acids, plant cells or other cells sensitive to osmotic shock and cells of bacteria, yeasts, viruses, mycoplasmas, protozoa, rickettsia, fungi and other small microbial cells and the like.

Target nucleic acids which are recognized by a plurality of multi-probes can be assayed to detect sequences which are present in less than 10% in a population of target nucleic acid

molecules, less than about 5%, less than about 1%, less than about 0.1%, and less than about 0.01% (e.g., such as specific gene sequences). The type of assay used to detect such sequences is a non-limiting feature of the invention and may comprise PCR or some other suitable assay as is known in the art or developed to detect recognition sequences which are found in less than 10% of a population of target nucleic acid molecules.

In one aspect, the assay to detect the less abundant recognition sequences comprises hybridizing at least one primer capable of specifically hybridizing to the recognition sequence but substantially incapable of hybridizing to more than about 50, more than about 25, more than about 10, more than about 5, more than about 2 target nucleic acid molecules (e.g., the probe recognizes both copies of a homozygous gene sequence), or more than one target nucleic acid in a population (e.g., such as an allele of a single copy heterozygous gene sequence present in a sample). In one preferred aspect a pair of such primers is provided and flank the recognition sequence identified by the multi-probe, i.e., are within an amplifiable distance of the recognition sequence such that amplicons of about 40-5000 bases can be produced, and preferably, 50-500 or more preferably 60-100 base amplicons are produced. One or more of the primers may be labelled.

Various amplifying reactions are well known to one of ordinary skill in the art and include, but are not limited to PCR, RT-PCR, LCR, in vitro transcription, rolling circle PCR, OLA and the like. Multiple primers can also be used in multiplex PCR for detecting a set of specific target molecules.

The invention further provides a method for designing multi-probes sequences for use in methods and kits according to the invention. A flow chart outlining the steps of the method is shown in Fig. 2.

In one aspect, a plurality of n-mers of n nucleotides is generated *in silico,* containing all possible n-mers. A subset of n-mers are selected which have a Tm $\geq$ 60°C. In another aspect, a subset of these probes is selected which do not self-hybridize to provide a list or database of candidate n-mers. The sequence of each n-mer is used to query a database comprising a plurality of target sequences. Preferably, the target sequence database comprises expressed sequences, such as human mRNA sequences.

From the list of candidate n-mers used to query the database, n-mers are selected that identify a maximum number of target sequences (e.g., n-mers which comprise recognition segments which are complementary to subsequences of a maximal number of target sequences in the target database) to generate an n-mer/target sequence matrix. Sequences of n-mers, which bind to a maximum number of target sequences, are stored in a database of optimal

probe sequences and these are subtracted from the candidate n-mer database. Target sequences that are identified by the first set of optimal probes are removed from the target sequence database. The process is then repeated for the remaining candidate probes until a set of multi-probes is identified comprising n-mers which cover more than about 60%, more than about 80%, more than about 90% and more than about 95% of targets sequences. The optimal sequences identified at each step may be used to generate a database of virtual multi-probes sequences. Multi-probes may then be synthesized which comprise sequences from the multi-probe database.

In another aspect, the method further comprises evaluating the general applicability of a given candidate probe recognition sequence for inclusion in the growing set of optimal probe candidates by both a query against the remaining target sequences as well as a query against the original set of target sequences. In one preferred aspect only probe recognition sequences that are frequently found in both the remaining target sequences and in the original target sequences are added to in the growing set of optimal probe recognition sequences. In a most preferred aspect this is accomplished by calculating the product of the scores from these queries and selecting the probes recognition sequence with the highest product that still is among the probe recognition sequences with 20% best score in the query against the current targets.

The invention also provides computer program products for facilitating the method described above (see, e.g., Fig. 2). In one aspect, the computer program product comprises program instructions, which can be executed by a computer or a user device connectable to a network in communication with a memory.

The invention further provides a system comprising a computer memory comprising a database of target sequences and an application system for executing instructions provided by the computer program product.

Kits Comprising Multi-Probes

A preferred embodiment of the invention is a kit for the characterisation or detection or quantification of target nucleic acids comprising samples of a library of multi-probes. In one aspect, the kit comprises in silico protocols for their use. In another aspect, the kit comprises information relating to suggestions for obtaining inexpensive DNA primers. The probes contained within these kits may have any or all of the characteristics described above. In one preferred aspect, a plurality of probes comprises a least one stabilizing nucleobase, such as an LNA nucleobase.

In another aspect, the plurality of probes comprises a nucleotide coupled or stably associated with at least one chemical moiety for increasing the stability of binding of the probe. In a further preferred aspect, the kit comprises a number of different probes for covering at least 60% of a population of different target sequences such as a transcriptome. In one preferred aspect, the transcriptome is a human transcriptome.

In another aspect, the kit comprises at least one probe labelled with one or more labels. In still another aspect, one or more probes comprise labels capable of interacting with each other in a FRET-based assay, i.e., the probes may be designed to perform in 5' nuclease or Molecular Beacon –based assays.

The kits according to the invention allow a user to quickly and efficiently to develop assays for many different nucleic acid targets. The kit may additionally comprise one or more reagents for performing an amplification reaction, such as PCR.

EXAMPLES

The invention will now be further illustrated with reference to the following examples. It will be appreciated that what follows is by way of example only and that modifications to detail may be made while still falling within the scope of the invention.

In the following Examples probe reference numbers designate the LNA-oligonucleotide sequences shown in the synthesis examples below.

EXAMPLE 1

Source of transcriptome data

The human transcriptome mRNA sequences were obtained from ENSEMBL. ENSEMBL is a joint project between EMBL - EBI and the Sanger Institute to develop a software system which produces and maintains automatic annotation on eukaryotic genomes (see, e.g., Butler, *Nature 406 (6794):* 333, 2000). ENSEMBL is primarily funded by the Wellcome Trust. It is noted that sequence data can be obtained from any type of database comprising expressed sequences, however, ENSEMBL is particularly attractive because it presents up-to-date sequence data and the best possible annotation for metazoan genomes. The file "Homo_sapiens.cdna.fa" was downloaded from the ENSEMBL ftp site: ftp://ftp.ensembl.org/pub/current_human/data/ on May 14. 2003. The file contains all EN-

SEMBL transcript predictions (i.e., 37347 different sequences). From each sequence the region starting at 50 nucleotides upstream from the 3' end to 1050 nucleotides upstream of the 3' end was extracted. The chosen set of probe sequences (see best mode below) was further evaluated against the human mRNA sequences in the Reference Sequence (RefSeq) collection from NCBI. RefSeq standards serve as the basis for medical, functional, and diversity studies; they provide a stable reference for gene identification and characterization, mutation analysis, expression studies, polymorphism discovery, and comparative analyses. The RefSeq collection aims to provide a comprehensive, integrated, non-redundant set of sequences, including genomic DNA, transcript (RNA), and protein products, for major research organisms. Similar coverage was found for both the 37347 sequences from ENSEMBL and the 19567 sequences in the RefSeq collection, i.e., demonstrating that the type of database is a non-limiting feature of the invention.

EXAMPLE 2

Calculation of a multi-probe dataset (Alfa library)

Special software running on UNIX computers was designed to calculate the optimal set of probes in a library. The algorithm is illustrated in the flow chart shown in Fig. 2.

The optimal coverage of a transcriptome is found in two steps. In the first step a sparse matrix of n_mers and genes is determined, so that the number of genes that contain a given n_mer can be found easily. This is done by running the getcover program with the -p option and a sequence file in FASTA format as input.

The second step is to determine the optimal cover with an algorithm, based on the matrix determined in the first step. For this purpose a program such as the getcover program is run with the matrix as input. However, programs performing similar functions and for executing similar steps may be readily designed by those of skill in the art.

Obtaining good oligonucleotide cover of the transcriptome.

1. All 4^n n-mers are generated and the expected melting temperature is calculated. n-mers with a melting temperature below 60°C or with high self-hybridisation energy are removed from the set. This gives a list of n-mers that have acceptable physical properties.
2. A list of gene sequences representing the human transcriptome is extracted from the ENSEMBL database.

3. Start of the main loop: Given the n-mer and gene list a sparse matrix of n-mers versus genes is generated by identifying all n-mers in a given gene and storing the result in a matrix.
4. If this is the first iteration, a copy of the matrix is put aside, and named the "total n-mer/gene matrix".
5. The n-mer that covers most genes is identified and the number of genes it covers is stored as "max_gene".
6. The coverage of the remaining genes in the matrix is determined and genes with coverage of at least 80% of max_gene are stored in the "n-mer list with good coverage".
7. The optimal n-mer is the one where the product of its current coverage and the total coverage is maximal.
8. The optimal n-mer is deleted from the n-mer list (step 1).
9. The genes covered by this n-mer are deleted from the gene list (step 2).
10. The n-mer is added to the optimal n-mer list, the process is continued from step 3 until no more n-mers can be found.

The program code ("getcover" version 1.0 by Niels Tolstrup 2003) for calculation of a multi-probe dataset is listed in Fig. 17. It consists of three proprietary modules: getcover.c, dyp.c, dyp.h

The program also incorporate four modules covered by the GNU Lesser General Public Licence:

```
getopt.c, getopt.h, getopt1.c, getopt_init.c
/* Copyright (C) 1987,88,89,90,91,92,93,94,95,96,98,99,2000,2001
Free Software Foundation, Inc.
These files are part of the GNU C Library. The GNU C Library is free software; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation */
```

The software was compiled with `aap`. The `main.aap` file used to make the program is likewise listed in Fig. 17.

To run the compiled program the following command is used:

```
getcover -l 9 -p -f < h_sap_cdna_50_1050.fasta > h_sap_cdna_50_1050_l9.stat
```

getcover -l 9 -i 1 -d 10 -t 60 -c -n -m -s < h_sap_cdna_50_1050_l9.stat > h_sap_cdna_50_1050_l9.cover

The computer program was used with instructions for implementing the algorithm described above to analyze the human transcriptome with the following parameter settings:

L89: probe length = 8 or 9 nucleotides
i1: inclusion fraction = 100%
d10: delta Tm required for target duplex against self duplex = 10°C
t60: minimum Tm for target duplex = 60 °C
c: complementary target sequence used as well
m80: optimal probes selected among the most general probes addressing the remaining targets with the product rule and the 80% rule
n: LNA nucleotides were preferably included in the central part of the recognition segment;

and resulted in the identification of a database of multi-probe target sequences.

Target sequences in this database are exemplary optimal targets for a multi-probe library. These optimal multi-probes are listed in TABLE 1 below and comprise 5' fluorescein fluorophores and 3' Eclipse quenchers (see below).

TABLE 1 Dual label oligonucleotide probes

cagcctcc	cagagcca	agctgtga	aggaggga
aggaggag	ctggaagc	cagagagc	tgtggaga
cccaggag	cagccaga	tgaggaga	ctggggaa
ctccagcc	cttctggg	acagtgga	ctcctgca
ctcctcca	ttctgcca	acagccat	tgaggtgg
ctgctgcc	aggagaga	tttctcca	aaggcagc
ctccagca	ttcctgca	cagtggtg	ctgtggca
ctgctggg	tttgggga	aaagggga	agaagggc
cttcctgg	caggcaga	tgtgggaa	tggatgga
acagcagc	ctgtgcca	actgggaa	ttctggca
cagctcca	ttccctgg	tcacagga	cagaaggc
ccccaccc	aaccccat	ttcctccc	atcccaga
tggtggtg	ctgcccag	aggtggaa	caggtgct
ttcctcca	ctgaggca	tgtggaca	ctgtctcc
ctgctcca	ctgctggt	tggaggcc	tgctgtga

tggagaga	cagtgcca	atggtgaa	agctggat
aaggcaga	atggggaa	ctggaagg	tggagagc
cagccagg	agggagag	caggcagc	cttggtgg
cagcagga	ctctgcca	tcaggagc	caccttgg
ctgtgctg	ctgctgag	acacacac	cagccacc
agaggaga	ccctccca	catcttca	ctgtgacc
ctgtggct	aggaggca	cacctgca	agggggaa
cagtggct	cactgcca	ccagggcc	tgggacca
ttctccca	ctgtgtgg	cagaggca	acagggaa
cctggagc	ttcccagt	ctgggact	ctgggcaa
cccagcag	tccagtgt	ctgcctgt	ctggagga
ttctcctg	ctcctccc	tggaaggc	tccactgc
cttcctgc	cttcccca	ctgtgcct	ctgccacc
ccacctcc	ctctgcca	ctgtgctc	acagcctca
ttcctctg	cagcaggt	ctgtgagc	ctgtggtc
tggtgatg	ctccatcc	tcctcctc	cttcaggc
tgtggctg	tgctgtcc	ctcagcca	tctgggtc
cttctccc	tcctctcc	ctcttccc	cttggagc
ctgcctcc	ctctgcct	ctgggcac	ccaggctc
ctccttcc	ctggctgc	tgggcatc	tctctggt
tcctgctc	ccgccgcc	ctctggct	cttgggct
catcctcc	ctcctcct	tgctgggc	ctgccatc
aggagctg	cagcctgg	ctgctctc	cactggga
tcctgctg	cagcagcc	ctggagtc	tgccctga
ctcctcca	tgctggag	cttcagcc	ttggtggt
ccagccag	cttcctcc	cttccagc	ttgggact
cagcccag	ttcctggc	tccaggtc	ctgctgga
ctccacca	tcctcagc	cagcatcc	caggagct
ctccagcc	aggagcag	cagaggct	ctcagcct
tggctctg	ccaggagg	ctgccttc	ttctggct
caggcagc	cagcctcc	ctgggaga	ctgtctgc
ctgcctct	agctggag	cccagccc	ctgtccca
cttctgcc	ctgctgcc	cagctccc	tctgccca
ctgctccc	tggctgtg	ccagccgc	ctggacac
tggtggaa	cctggaga	cctcagcc	ttgccatc
agctggga	ccagggcc	tcctcttct	cttcccct
ctgcttcc	ccaccacc	ctggctcc	cttgggca
cagcaggc	tctgctgc	ccagggca	ttctggtc

tctggagc	cagccacc	ctccacct	ccgccgcc
catccagc	cagaggag	ctgcccca	cttcttctc
atggctgc	ctctcctc	tgggcagc	ttccctcc
ctcctgcc	caggagcc	ctggtctc	ttcctcaga
tggtggcc	tctggtcc	ctggggcc	tccaaggc
ctggggct	ctgtctcc	cagtggca	ttggggtc

These hyper-abundant 9-mer and 8-mer sequences fulfil the selection criteria in Fig. 2., i.e.,

- each probe target occurs in at least 6% of the sequences in the human transcriptome (i.e., more than 2200 target sequences each, more than 800 sequences targeted within 1000 nt proximal to the 3' end of the transcript).
- they are not self complementary (i.e. unlikely to form probe duplexes). Self score is at least 10 below T_m estimate for the duplex formed with the target.
- the formed duplex with their target sequence has a T_m at or above 60 °C.

They cover > 98 % of the mRNAs in the human transcriptome when combined.

Especially preferred versions of the multi-probes of table 1 are presented in the following table 1a:

TABLE 1a One hundred LNA substituted oligonucleotides

cAgCCTCc	cAGAGCCa	aGCTGTGa	aGGAGGGa
aGGAGGAg	cTGGAAGc	cAGAGAGc	tGTGGAGa
ccCAGGAg	cAGCCAGa	tGAGGAGa	ctGGGGAa
cTCCAgCc	cTTCTGGg	aCAGTGGa	cTCCtGCa
cTCCTCCa	tTCTGCCa	aCAGCCAt	tGAGGtGg
cTgCTGCc	aGGAGAGa	tTTCTCCa	aAGGCAGc
cTCCAGCa	tTCCTGCa	cAGTGGTg	ctGTGGCa
cTGCTGgg	tTTGGGGa	aAAGGGGa	aGAAGGGc
cTTCCTGg	cAGGCAGa	tGTGGGAa	tGGATGGa
aCAGCAGc	ctGTGCCa	aCTGGGAa	tTCTGGCa
caGCTCCa	tTCCCTGg	tCACAGGa	cAGAAGGc
cCCCACCc	aACCCCAt	tTCCTCCc	aTCCCAGa
tGGTGGTg	ctGCCCag	aGGTGGAa	cAGGtGCt
tTCCTCCa	cTGAGGCa	tGTGGACa	cTGTCTCc
cTGCTCCa	cTGCtGGt	tGGAGgCc	tGCTGTGa

tGGAGAGa	cAGtGCCa	atGGTGAA	aGCTGGAt
aAGGCAGa	aTGGGGAa	cTGGAAGg	tGGAGAGc
cAGCcAGg	aGGGAGAg	cAGGcAGc	cTTGGTGg
cAGCAGGa	cTCtGCCa	tCAGGaGc	cACCTTGg
cTGTGCTg	cTGCTGAg	aCACACAC	cAgCCACc
aGAGGAGa	cCCtCCCa	cATCTTCA	cTGTGACc
ctGTGGCt	aGGAGGca	cACCtGCa	aGGGGGAa
caGTGGCt	cACtGCCa	cCAGgGcc	tGgGACCa
tTCTCCCa	cTGTGTGg	cAGAGGCa	aCAGGGAa

- wherein small letters designate a DNA monomer and capital letters designate a LNA nucleotide.

> 95.0 % of the mRNA sequences are targeted within the 1000 nt near their 3' terminal, (position 50 to 1050 from 3' end) and > 95% of the mRNA contain the target sequence for more than one probe in the library. More than 650,000 target sites for these 100 multi-probes were identified in the human transcriptome containing 37,347 nucleic acid sequences. The average number of multi-probes addressing each transcript in the transcriptome is 17.4 and the median value is target sites for 14 different probes.

The sequences noted above are also an excellent choice of probes for other transcriptomes, though they were not selected to be optimized for the particular organisms. We have thus evaluated the coverage of the above listed library for the mouse and rat genome despite the fact that the above probes were designed to detect/characterize/quantify the transcripts in the human transcriptome only. E.g. see table 2.

TABLE 2	Transcriptome		
Human probe library	Human	Mouse	Rat
no. of mRNA sequences	37347	32911	28904
Coverage of full length mRNAs	**96.7%**	**94.6%**	**93.5%**
Coverage 1000 nt near the 3'-end	**91.0%**	-	-
At least covered by two probes	**89.8%**	**80.2%**	**77.0%**

nt ~ nucleotides.

EXAMPLE 3

Expected coverage of human transcriptome by frequently occurring 9-mer oligonucleotides

Experimental pilot data (similar to Fig. 6) indicated that it is possible to reduce the length of the recognition sequence of a dual-labelled probe for real-time PCR assays to 8 or 9 nucleotides depending on the sequence, if the probe is enhanced with LNA. The unique duplex stabilizing properties of LNA are necessary to ensure an adequate stability for such a short duplex (i.e. T_m > 60 °C). The functional real-time PCR probe will be almost pure LNA with 6 to 10 LNA nucleotides in the recognition sequence. However, the short recognition sequence makes it possible to use the same LNA probe to detect and quantify the abundance of many different genes. By proper selection of the best (i.e. most common) 8 or 9-mer recognition sequences according to the algorithm depicted in Fig. 2 it is possible to get a coverage of the human transcriptome containing about 37347 mRNAs (Fig. 3).

Fig. 3 shows the expected coverage as percentage of the total number of mRNA sequences in the human transcriptome that are detectable within a 1000 nt long stretch near the 3' end of the respective sequences (i.e. the sequence from 50 nt to 1050 nt from the 3' end) by optimized probes of different lengths. The probes are required to be sufficiently stable (Tm>60 degC) and with a low propensity for forming self duplexes, which eliminate many 9-mers and even more 8-mer probe sequences.

If all probes sequences of a given length could be used as probes we would obviously get the best coverage of the transcriptome by the shortest possible probe sequences. This is indeed the case when only a limited number of probes (< 55) are included in the library (Fig. 4). However, because many short probes with a low GC content have an inadequate thermal stability, they were omitted from the library. The limited diversity of acceptable 8-mer probes are less efficient at detecting low GC content genes, and a library composed of 100 different 9-mer probes consequently have a better coverage of the transcriptome than a similar library of 8-mers. However, the best choice is a mixed library composed of sequences of different lengths such as the proposed best mode library listed above. The coverage of this library is not shown in Fig. 4.

The designed probe library containing 100 of the most commonly occurring 9-mer and 8-mers, i.e., the "Human mRNA probe library" can be handled in a convenient box or microtiter plate format.

The initial set of 100 probes for human mRNAs can be modified to generate similar library kits for transcriptomes from other organisms (mouse, rat, *Drosophila*, *C. elegans*, yeast,

Arabidopsis, zebrafish, primates, domestic animals, etc.). Construction of these new probe libraries will require little effort, as most of the human mRNA probes may be re-used in the novel library kits (TABLE 2).

EXAMPLE 4

Number of probes in the library that target each gene

Not only does the limited number of probes in the proposed libraries target a large fraction (> 98%) of the human transcriptome, but there is also a large degree of redundancy in that most of the genes (almost 95%) may be detected by more than one probe. More than 650,000 target sites have been identified in the human transcriptome (37347 genes) for the 100 probes in the best mode library shown above. This gives an average number of target sites per probe of 6782 (i.e. 18 % of the transcriptome) ranging from 2527 to 12066 sequences per probe. The average number of probes capable of detecting a particular gene is 17.4, and the median value is 14. Within the library of only 100 probes we thus have at least 14 probes for more than 50% of all human mRNA sequences.

The number of genes that are targeted by a given number of probes in the library is depicted in Fig. 4.

EXAMPLE 5

Design of 9-mer probes to demonstrate feasibility

The SSA4 gene from yeast (*Saccharomyces cerevisiae*) was selected for the expression assays because the gene transcription level can be induced by heat shock and mutants are available where expression is knocked out. Three different 9mer sequences were selected amongst commonly occurring 9mer sequences within the human transcriptome (Table 3). The sequences were present near the 3' terminal end of 1.8 to 6.4 % of all mRNA sequences within the human transcriptome. Further selection criteria were a moderate level of selfcomplementarity and a Tm of 60°C or above. All three sequences were present within the terminal 1000 bases of the SSA4 ORF. Three 5' nuclease assay probes were constructed by synthesizing the three sequences with a FITCH fluorophore in the 5'-end and an Eclipse quencher (Epoch Biosciences) in the 3'end. The probes were named according to their position within the ORF YER103W (SSA4) where position 1201 was set to be position 1. Three sets of primer pairs were designed to produce three non-overlapping amplicons, which each contained one

of the three probe sequences. Amplicons were named according to the probe sequence they encompassed.

Table 3. Designed 5′ nuclease assay probes and primers

Sequence	Name of probe	Forward primer sequence	Reverse primer sequence	Amplicon length
aaGGAGAAG	Dual-labelled-469	cgcgtttactttgaaaaatt ctg (SEQ ID NO: 1)	gcttccaatttcctggca tc (SEQ ID NO: 2)	81 bp
cAAGGAAAg	Dual-labelled-570	gcccaagatgctataaatt-ggttag (SEQ ID NO: 3)	gggtttgcaacaccttct agttc (SEQ ID NO: 4)	95 bp
ctGGAGCaG	Dual-labelled-671	tacggagctgcaggtggt (SEQ ID NO: 5)	gttgggccgttgtctggt (SEQ ID NO: 6)	86 bp

bp ~ base pairs

Two Molecular Beacons were also designed to detect the SSA4 469- and the SSA4 570 sequence and named Beacon-469 and Beacon-570, respectively. The sequence of the SSA4 469 beacon was CAAGGAGAAGTTG (SEQ ID NO: 7, 10-mer recognition site) which should enable this oligonucleotide to form the intramolecular beacon structure with a stem formed by the LNA-LNA interactions between the 5′-CAA and the TTG-3′. The sequence of the SSA4 570 beacon was CAAGGAAAGttG (9-mer recognition site) where the intramolecular beacon structure may form between the 5′-CAA and the ttG-3′. Both the sequences were synthesized with a fluorescein fluorophore in the 5′-end and a Dabcyl quencher in the 3′end.

One SYBR Green labeled probe was also designed to detect the SSA4 570 sequence and named SYBR-Probe-570. The sequence of this probe was CAAGGAAaG. This probe was synthesized with a amino-C6 linker on the 5′-end on which the fluorophore SYBR Green 101 (Molecular Probes) was attached according to the manufactures instructions. Upon hybridization to the target sequence, the linker attached fluorophore should intercalate in the generated LNA-DNA duplex region causing increased fluorescence from the SYBR Green 101.

TABLE 4: SEQUENCES

EQ Number	Name	Type	Sequence	Position in gene
13992	Dual-labelled-469	5′ nuclease assay probe	5'-Fluor-aaGGAGAAG-Eclipse-3'	469-47
13994	Dual-labelled-	5′ nuclease assay probe	5'-Fluor-cAAGGAAAg-Eclipse-3'	570-57

EQ Number	Name	Type	Sequence	Position in gene
13996	570 Dual-labelled-671	5' nuclease assay probe	5'-Fluor-ctGGAGCaG-Eclipse-3'	671-67
13997	Beacon-469	Molecular Beacon	5'-Fluor-CAAGGAGAAGTTG-Dabcyl-3' (5'-Fluor-SEQ ID NO: 8-Dabcyl-3')	
14148	Beacon-570	Molecular Beacon	5'-Fluor-CAAGGAAAGttG-Dabcyl-3' (5'-Fluor-SEQ ID NO: 9-Dabcyl-3')	
14165	SYBR-Probe-570	SYBR-Probe	5'-SYBR101-NH2C6-cAAGGAAAg-3'	
14012	SSA4-469-F	Primer	cgcgtttactttgaaaaattctg (SEQ ID NO: 10)	
14013	SSA4-469-R	Primer	gcttccaatttcctggcatc (SEQ ID NO: 11)	
14014	SSA4-570-F	Primer	gcccaagatgctataaattggttag (SEQ ID NO: 12)	
14015	SSA4-570-R	Primer	gggtttgcaacaccttctagttc (SEQ ID NO: 13)	
14016	SSA4-671-F	Primer	tacggagctgcaggtggt (SEQ ID NO: 14)	
14017	SSA4-671-R	Primer	gttgggccgttgtctggt (SEQ ID NO: 15)	
14115	POL5-469-F	Primer	gcgagagaaaacaagcaagg (SEQ ID NO: 16)	
14116	POL5-469-R	Primer	attcgtcttcactggcatca (SEQ ID NO: 17)	
14117	APG9-570-F	Primer	cagctaaaaatgatgacaataatgg (SEQ ID NO: 18)	
14118	APG9-570-R	Primer	attacatcatgattagggaatgc (SEQ ID NO: 19)	
14119	HSP82-671-F	Primer	gggtttgaacattgatgagga (SEQ ID NO: 20)	
14120	HSP82-671-R	Primer	ggtgtcagctggaacctctt (SEQ ID NO: 21)	

EXAMPLE 6

Synthesis, deprotection and purification of dual labelled oligonucleotides

The dual labelled oligonucleotides EQ13992 to EQ14148 (Table 4) were prepared on an automated DNA synthesizer (Expedite 8909 DNA synthesizer, PerSeptive Biosystems, 0.2 μmol scale) using the phosphoramidite approach (Beaucage and Caruthers, *Tetrahedron Lett. 22*: 1859-1862, 1981) with 2-cyanoethyl protected LNA and DNA phosphoramidites, (Sinha, et al., *Tetrahedron Lett.24*: 5843-5846, 1983). CPG solid supports derivatized with either

eclipse quencher (EQ13992-EQ13996) or dabcyl (EQ13997-EQ14148) and 5'-fluorescein phosphoramidite (GLEN Research, Sterling, Virginia, USA). The synthesis cycle was modified for LNA phosphoramidites (250s coupling time) compared to DNA phosphoramidites. 1*H*-tetrazole or 4,5-dicyanoimidazole (Proligo, Hamburg, Germany) was used as activator in the coupling step.

The oligonucleotides were deprotected using 32% aqueous ammonia (1h at room temperature, then 2 hours at 60°C) and purified by HPLC (Shimadzu-SpectraChrom series; Xterra™ RP18 column, 10?m 7.8 x 150 mm (Waters). Buffers: A: 0.05M Triethylammonium acetate pH 7.4. B. 50% acetonitrile in water. Eluent: 0-25 min: 10-80% B; 25-30 min: 80% B). The composition and purity of the oligonucleotides were verified by MALDI-MS (PerSeptive Biosystem, Voyager DE-PRO) analysis, see Table 5. Fig. 5 is the MALDI-MS spectrum of EQ13992 showing [M-H]- = 4121,3 Da. This is a typical MALDI-MS spectrum for the 9-mer probes of the invention.

TABLE 5:

EQ#	Sequences	MW (Calc.)	MW (Found)
13992	5'-Fitc-aaGGAGAAG-*EQL*-3'	4091,8 Da.	4091,6 Da.
13994	5'-Fitc-cAAGGAAAg-*EQL*-3'	4051,9 Da.	4049,3 Da.
13996	5'-Fitc-ctGGAGmCaG-*EQL*-3'	4020,8 Da.	4021,6 Da.
13997	5'-Fitc-mCAAGGAGAAGTTG-*dabcyl*-3' (5'-Fitc-SEQ ID NO: 22-*dabcyl*-3')	5426,3 Da.	5421,2 Da.

Capitals designate LNA monomers (A, G, mC, T), where mC is LNA methyl cytosine. Small letters designate DNA monomers (a, g, c, t). Fitc = Fluorescein; EQL = Eclipse quencher; Dabcyl = Dabcyl quencher. MW = Molecular weight.

EXAMPLE 7

Production of cDNA standards of SSA4 for detection with 9-mer probes

The functionality of the constructed 9mer probes were analysed in PCR assays where the probes ability to detect different SSA4 PCR amplicons were questioned. Template for the PCR reaction was cDNA obtained from reverse transcription of cRNA produced from in vitro tran-

scription of a downstream region of the SSA4 gene in the expression vector pTRIamp18 (Ambion). The downstream region of the SSA4 gene was cloned as follows:

PCR amplification

Amplification of the partial yeast gene was done by standard PCR using yeast genomic DNA as template. Genomic DNA was prepared from a wild type standard laboratory strain of *Saccharomyces cerevisiae* using the Nucleon MiY DNA extraction kit (Amersham Biosciences) according to supplier's instructions. In the first step of PCR amplification, a forward primer containing a restriction enzyme site and a reverse primer containing a universal linker sequence were used. In this step 20 bp was added to the 3'-end of the amplicon, next to the stop codon. In the second step of amplification, the reverse primer was exchanged with a nested primer containing a poly-T_{20} tail and a restriction enzyme site. The SSA4 amplicon contains 729 bp of the *SSA4* ORF plus a 20 bp universal linker sequence and a poly-A_{20} tail.

The PCR primers used were:

YER103W-For-SacI: acgtgagctcattgaaactgcaggtggtattatga (SEQ ID NO: 23)

YER103W-Rev-Uni: gatccccgggaattgccatgctaatcaacctcttcaaccgttgg (SEQ ID NO: 24)

Uni-polyT-BamHI: acgtggatccttttttttttttttttttttttgatccccgggaattgccatg (SEQ ID NO: 25).

Plasmid DNA constructs

The PCR amplicon was cut with the restriction enzymes, *Eco*RI + *Bam*HI. The DNA fragment was ligated into the pTRIamp18 vector (Ambion) using the Quick Ligation Kit (New England Biolabs) according to the supplier's instructions and transformed into *E. coli* DH-5 by standard methods.

DNA sequencing

To verify the cloning of the PCR amplicon, plasmid DNA was sequenced using M13 forward and M13 reverse primers and analysed on an ABI 377.

In vitro transcription

SSA4 cRNA was obtained by performing in vitro transcription with the Megascript T7 kit (Ambion) according to the supplier's instructions.

Reverse transcription

Reverse transcription was performed with 1µg of cRNA and 0.2 U of the reverse transcriptase Superscript II RT (Invitrogen) according to the suppliers instructions except that 20 U Superase-In (RNAse inhibitor - Ambion) was added. The produced cDNA was purified on a QiaQuick PCR purification column (Qiagen) according to the supplier's instructions using the supplied EB-buffer for elution. The DNA concentration of the eluted cDNA was measured and diluted to a concentration of SSA4 cDNA copies corresponding to 2×10^7 copies pr µL.

EXAMPLE 8

Protocol for of dual label probe assays

Reagents for the dual label probe PCRs were mixed according to the following scheme (Table 6):

Table 6

Reagents	**Final Concentration**
H_2O	
GeneAmp 10x PCR buffer II	1x
Mg^{2+}	5.5 mM
DNTP	0.2 mM
Dual Label Probe	0.1 or 0.3 µM*
Template	1 µL
Forward primer	0.2 µM
Reverse primer	0.2 µM
AmliTaq Gold	2.5 U
Total	50 µL

*) Final concentration of 5' nuclease assay probe 0.1 µM and Beacon/SYBR-probe 0.3 µM.

In the present experiments 2 x 10^7 copies of the SSA4 cDNA was added as template. Assays were performed in a DNA Engine Opticon® (MJ Research) using the following PCR cycle protocols:

Table 7

5′ nuclease assays	Beacon & SYBR-probe Assays
95°C for 7 minutes	95°C for 7 minutes
&	&
40 cycles of:	40 cycles of:
94°C for 20 seconds	94°C for 30 seconds
60°C for 1 minute	52°C for 1 minute*
Fluorescence detection	**Fluorescence detection**
	72°C for 30 seconds

* For the Beacon-570 with 9-mer recognition site the annealing temperature was reduced to 44°C

The composition of the PCR reactions shown in Table 6 together with PCR cycle protocols listed in Table 7 will be referred to as standard 5′ nuclease assay or standard Beacon assay conditions.

EXAMPLE 9

Specificity of 9-mer 5′ nuclease assay probes

The specificity of the 5′ nuclease assay probes were demonstrated in assays where each of the probes was added to 3 different PCR reactions each generating a different SSA4 PCR amplicon. As shown in Fig. 6, each probe only produces a fluorescent signal together with the amplicon it was designed to detect (see also Figs. 10, 11 and 12). Importantly the different probes had very similar cycle threshold C_t values (from 23.2 to 23.7), showing that the assays and probes have a very equal efficiency. Furthermore it indicates that the assays should detect similar expression levels when used in used in real expression assays. This is an important finding, because variability in performance of different probes is undesirable.

EXAMPLE 10

Specificity of 9 and 10-mer Molecular Beacon probes

The ability to detect in real time, newly generated PCR amplicons was also demonstrated for the molecular beacon design concept. The Molecular Beacon designed against the 469 amplicon with a 10-mer recognition sequence produced a clear signal when the SSA4 cDNA template and primers for generating the 469 amplicon were present in the PCR, Fig. 7A. The observed C_t value was 24.0 and very similar to the ones obtained with the 5' nuclease assay probes again indicating a very similar sensitivity of the different probes. No signal was produced when the SSA4 template was not added. A similar result was produced by the Molecular Beacon designed against the 570 amplicon with a 9-mer recognition sequence, Fig. 7B.

EXAMPLE 11.

Specificity of 9-mer SYBR-probes.

The ability to detect newly generated PCR amplicons was also demonstrated for the SYBR-probe design concept. The 9-mer SYBR-probe designed against the 570 amplicon of the SSA4 cDNA produced a clear signal when the SSA4 cDNA template and primers for generating the 570 amplicon were present in the PCR, Fig. 8. No signal was produced when the SSA4 template was not added.

EXAMPLE 12

Quantification of transcript copy number

The ability to detect different levels of gene transcripts is an essential requirement for a probe to perform in a true expression assay. The fulfilment of the requirement was shown by the three 5' nuclease assay probes in an assay where different levels of the expression vector derived SSA4 cDNA was added to different PCR reactions together with one of the 5' nuclease assay probes (Fig. 9). Composition and cycle conditions were according to standard 5' nuclease assay conditions.

The cDNA copy number in the PCR before start of cycling is reflected in the cycle threshold value C_t, i.e., the cycle number at which signal is first detected. Signal is here only defined as signal if fluorescence is five times above the standard deviation of the fluorescence detected

in PCR cycles 3 to 10. The results show an overall good correlation between the logarithm to the initial cDNA copy number and the C_t value (Fig. 9). The correlation appears as a straight line with slope between -3.456 and -3.499 depending on the probe and correlation coefficients between 0.9981 and 0.9999. The slope of the curves reflect the efficiency of the PCRs with a 100% efficiency corresponding to a slope of -3.322 assuming a doubling of amplicon in each PCR cycle. The slopes of the present PCRs indicate PCR efficiencies between 94% and 100%. The correlation coefficients and the PCR efficiencies are as high as or higher than the values obtained with DNA 5' nuclease assay probes 17 to 26 nucleotides long in detection assays of the same SSA4 cDNA levels (results not shown). Therefore these result show that the three 9-mer 5' nuclease assay probes meet the requirements for true expression probes indicating that the probes should perform in expression profiling assays

EXAMPLE 13

Detection of SSA4 transcription levels in yeast

Expression levels of the SSA4 transcript were detected in different yeast strains grown at different culture conditions (± heat shock). A standard laboratory strain of *Saccharomyces cerevisiae* was used as wild type yeast in the experiments described here. A SSA4 knockout mutant was obtained from EUROSCARF (accession number Y06101). This strain is here referred to as the SSA4 mutant. Both yeast strains were grown in YPD medium at 30°C till an OD_{600} of 0.8 A. Yeast cultures that were to be heat shocked were transferred to 40°C for 30 minutes after which the cells were harvested by centrifugation and the pellet frozen at -80°C. Non-heat shocked cells were in the meantime left growing at 30°C for 30 minutes and then harvested as above.

RNA was isolated from the harvested yeast using the FastRNA Kit (Bio 101) and the FastPrep machine according to the supplier's instructions.

Reverse transcription was performed with 5 µg of anchored oligo(dT) primer to prime the reaction on 1µg of total RNA, and 0.2 U of the reverse transcriptase Superscript II RT (Invitrogen) according to the suppliers instructions except that 20 U Superase-In (RNAse inhibitor - Ambion) was added. After a two-hour incubation, enzyme inactivation was performed at 70° for 5 minutes. The cDNA reactions were diluted 5 times in 10 mM Tris buffer pH 8.5 and oligonucleotides and enzymes were removed by purification on a MicroSpin™ S-400 HR column (Amersham Pharmacia Biotech). Prior to performing the expression assay the cDNA was diluted 20 times. The expression assay was performed with the Dual-labelled-570 probe using standard 5' nuclease assay conditions except 2 µL of template was added. The template was

a 100 times dilution of the original reverse transcription reactions. The four different cDNA templates used were derived from wild type or mutant with or without heat shock. The assay produced the expected results (Fig. 10) showing increased levels of the SSA4 transcript in heat shocked wild type yeast (C_t =26.1) compared to the wild type yeast that was not submitted to elevated temperature (C_t =30.3). No transcripts were detected in the mutant yeast irrespective of culture conditions. The difference in C_t values of 3.5 corresponds to a 17 fold induction in the expression level of the heat shocked versus the non-heat shocked wild type yeast and this value is close to the values around 19 reported in the literature (Causton, et al. 2001). These values were obtained by using the standard curve obtained for the Dual-labelled-570 probe in the quantification experiments with known amounts of the SSA4 transcript (see Fig. 9). The experiments demonstrate that the 9-mer probes are capable of detecting expression levels that are in good accordance with published results.

EXAMPLE 14

Multiple transcript detection with individual 9-mer probes

To demonstrate the ability of the three 5′ nuclease assay probes to detect expression levels of other genes as well, three different yeast genes were selected in which one of the probe sequences was present. Primers were designed to amplify a 60-100 base pair region around the probe sequence. The three selected yeast genes and the corresponding primers are shown in Table.

TABLE 8

Design of alternative expression assays

Sequence/Name	Matching Probe	Forward primer sequence	Reverse primer sequence	Amplicon length
YEL055C/POL5	Dual-labelled-469	gcgagagaaaaca-agcaagg (SEQ ID NO: 26)	attcgtcttcactggcatca (SEQ ID NO: 27)	94 bp
YDL149W_APG9	Dual-labelled-570	cagctaaaaatgat-gacaataatgg (SEQ ID NO: 28)	attacatcatgattaggga-atgc (SEQ ID NO: 29)	97 bp
YPL240C_HSP82	Dual-labelled-671	gggtttgaacattg-atgagga (SEQ ID NO: 30)	ggtgtcagctggaacctctt (SEQ ID NO: 31)	88 bp

Total cDNA derived from non-heat shocked wild type yeast was used as template for the expression assay, which was performed using standard 5' nuclease assay conditions except 2 µL of template was added. As shown in Fig. 11, all three probes could detect expression of the genes according to the assay design outlined in Table 8. Expression was not detected with any other combination of probe and primers than the ones outlined in Table 8. Expression data are available in the literature for the SSA4, POL5, HSP82, and the APG9 (Holstege, et al. 1998). For non-heat shocked yeast, these data describe similar expression levels for SSA4 (0.8 transcript copies per cell), POL5 (0.8 transcript copies per cell) and HSP82 (1.3 transcript copies per cell) whereas APG9 transcript levels are somewhat lower (0.1 transcript copies per cell).

This data is in good correspondence with the results obtained here since all these genes showed similar C_t values except HSP82, which had a C_t value of 25.6. This suggests that the HSP82 transcript was more abundant in the strain used in these experiments than what is indicated by the literature. Agarose gel electrophoresis was performed with the PCRs shown in Fig. 11a for the Dual-labelled-469 probe. The agarose gel (Fig. 12) shows that PCR product was indeed generated in reactions where no signal was obtained and therefore the lack fluorescent signal from these reactions was not caused by failure of the PCR. Furthermore, the different length of amplicons produced in expression assays for different genes indicate that the signal produced in expression assays for different genes are indeed specific for the gene in question.

EXAMPLE 15

Selection of targets

Using the EnsMart software release 16.1 from http://www.ensembl.org/EnsMart, the 50 bases from each end off all exons from the Homo Sapiens NCBI 33 dbSNP115 Ensembl Genes were extracted to form a Human Exon50 target set. Using the GetCover program (cf. Fig. 17), occurrence of all probe target sequences was calculated and probe target sequences not passing selection criteria according to excess self-complimentarity, excessive GC content etc. were eliminated. Among the remaining sequences, the most abundant probe target sequences was selected (No. 1, covering 3200 targets), and subsequently all the probe targets having a prevalence above 0.8 times the prevalence of the most abundant (3200 x 0.8) or above 2560 targets. From the remaining sample the number of new hits for each probe was computed and the product of number of new hits per probe target compared to the existing selection and the total prevalence of the same probe target was computed and used to select the next most abundant probe target sequence by selecting the highest product number. The

probe target length (n), and sequence (nmer) and occurrence in the total target (cover), as well as the number of new hits per probe target selection (Newhit), the product of Newhit and cover (newhit x cover) and the number of accumulated hits in the target population from all accumulated probes (sum) is exemplified in the table below.

No	n	nmer	Newhit	Cover	newhit x cover	sum
1	8	ctcctcct	3200	3200	10240000	3200
2	8	ctggagga	2587	3056	7905872	5787
3	8	aggagctg	2132	3074	6553768	7919
4	8	cagcctgg	2062	2812	5798344	9981
5	8	cagcagcc	1774	2809	4983166	11755
6	8	tgctggag	1473	2864	4218672	13228
7	8	agctggag	1293	2863	3701859	14521
8	8	ctgctgcc	1277	2608	3330416	15798
9	8	aggagcag	1179	2636	3107844	16977
10	8	ccaggagg	1044	2567	2679948	18021
11	8	tcctgctg	945	2538	2398410	18966
12	8	cttcctcc	894	2477	2214438	19860
13	8	ccgccgcc	1017	2003	2037051	20877
14	8	cctggagc	781	2439	1904859	21658
15	8	cagcctcc	794	2325	1846050	22452
16	8	tggctgtg	805	2122	1708210	23257
17	8	cctggaga	692	2306	1595752	23949
18	8	ccagccag	661	2205	1457505	24610
19	8	ccagggcc	578	2318	1339804	25188
20	8	cccagcag	544	2373	1290912	25732
21	8	ccaccacc	641	1916	1228156	26373
22	8	ctcctcca	459	3010	1381590	26832
23	8	ttctcctg	534	1894	1011396	27366
24	8	cagcccag	471	2033	957543	27837
25	8	ctggctgc	419	2173	910487	28256
26	8	ctccacca	426	2097	893322	28682
27	8	cttcctgc	437	1972	861764	29119
28	8	cttccagc	415	1883	781445	29534
29	8	ccacctcc	366	2018	738588	29900
30	8	ttcctctg	435	1666	724710	30335
31	8	cccagccc	354	1948	689592	30689

No	n	nmer	Newhit	Cover	newhit x cover	sum
32	8	tggtgatg	398	1675	666650	31087
33	8	tggctctg	358	1767	632586	31445
34	8	ctgccttc	396	1557	616572	31841
35	8	ctccagcc	294	2378	699132	32135
36	8	tgtggctg	304	1930	586720	32439
37	8	cagaggag	302	1845	557190	32741
38	8	cagctccc	275	1914	526350	33016
39	8	ctgcctcc	262	1977	517974	33278
40	8	tctgctgc	267	1912	510504	33545
41	8	ctgcttcc	280	1777	497560	33825
42	8	cttctccc	291	1663	483933	34116
43	8	cctcagcc	232	1863	432216	34348
44	8	ctccttcc	236	1762	415832	34584
45	8	cagcaggc	217	1868	405356	34801
46	8	ctgcctct	251	1575	395325	35052
47	8	ctccacct	215	1706	366790	35267
48	8	ctcctccc	205	1701	348705	35472
49	8	cttcccca	224	1537	344288	35696
50	8	cttcagcc	203	1650	334950	35899
51	8	ctctgcca	201	1628	327228	36100
52	8	ctgggaga	192	1606	308352	36292
53	8	cttctgcc	195	1533	298935	36487
54	8	cagcaggt	170	1711	290870	36657
55	8	tctggagc	206	1328	273568	36863
56	8	tcctgctc	159	1864	296376	37022
57	8	ctggggcc	159	1659	263781	37181
58	8	ctcctgcc	155	1733	268615	37336
59	8	ctgggcaa	185	1374	254190	37521
60	8	ctggggct	149	1819	271031	37670
61	8	tggtggcc	145	1731	250995	37815
62	8	ccagggca	147	1613	237111	37962
63	8	ctgctccc	146	1582	230972	38108
64	8	tgggcagc	135	1821	245835	38243
65	8	ctccatcc	161	1389	223629	38404
66	8	ctgcccca	143	1498	214214	38547
67	8	ttcctggc	155	1351	209405	38702

No	n	nmer	Newhit	Cover	newhit x cover	sum
68	8	atggctgc	157	1285	201745	38859
69	8	tggtggaa	155	1263	195765	39014
70	8	tgctgtcc	135	1424	192240	39149
71	8	ccagccgc	159	1203	191277	39308
72	8	catccagc	122	1590	193980	39430
73	8	tcctctcc	118	1545	182310	39548
74	8	agctggga	121	1398	169158	39669
75	8	ctggtctc	128	1151	147328	39797
76	8	ttcccagt	142	1023	145266	39939
77	8	caggcagc	108	1819	196452	40047
78	8	tcctcagc	105	1654	173670	40152
79	8	ctggctcc	103	1607	165521	40255
80	9	tcctcttct	127	1006	127762	40382
81	8	tccagtgt	123	968	119064	40505

EXAMPLE 16

qPCR for Human Genes

Use of the Probe library is coupled to the use of a real-time PCR design software which can:

- recognise an input sequence via a unique identifier or by registering a submitted nucleic acid sequence
- identify all probes which can target the nucleic acid
- sort probes according to target sequence selection criteria such as proximity to the 3' end or proximity to intron-exon boundaries
- if possible, design PCR primers that flank probes targeting the nucleic acid sequence according to PCR design rules
- suggest available real-time PCR assays based on above procedures.

The design of an efficient and reliable qPCR assay for a human gene is carried out via the software found on www.probelibrary.com

The ProbeFinder software designs optimal qPCR probes and primers fast and reliably for a given human gene.

The design comprises the following steps:

1) Determination of the intron positions
Noise from chromosomal DNA is eliminated by selecting intron spanning qPCR's. Introns are determined by a blast search against the human genome. Regions found on the DNA, but not in the transcript are considered to be introns.

2) Match of the Probe Library to the gene
Virtually all human transcripts are covered by at least one of the 90 probes, the high coverage is made possible by LNA modifications of the recognition sequence tags.

3) Design of primers and selection of optimal qPCR assay
Primers are designed with 'Primer3' (Whitehead Inst. For Biomedical Research, S. Rozen and H.J. Skaletsky). Finally the probes are ranked according to selected rules ensuring the best possible qPCR. The rules favor intron spanning amplicons to remove false signals from DNA contamination, small amplicon size for reproducible and comparable assays and a GC content optimized for PCR.

EXAMPLE 17

Preparation of ena-monomers and oligomers

ENA-T monomers are prepared and used for the preparation of dual labelled probes of the invention.
In the following sequences the X denotes a 2'-O,4'-C-ethylene-5-methyluridine (ENA-T). The synthesis of this monomer is described in WO 00/47599. The reaction conditions for incorporation of a 5'-O-Dimethoxytrityl-2'-O,4'-C-ethylene-5-methyluridine-3'-O-(2-cyanoethyl-N,N-diisopropyl)phosphoramidite corresponds to the reaction conditions for the preparation of LNA oligomers as described in EXAMPLE 6.

The following three dual labelled probes are preparted:

EQ#	Sequences	MW (Calc.)	MW (Found)
16533	5'-Fitc-ctGmCXmCmCAg-*EQL*-3'	4002 Da.	4001 Da.
16534	5'-Fitc-cXGmCXmCmCA-*EQL*-3'	3715 Da.	3716 Da.
16535	5'-Fitc-tGGmCGAXXX-*EQL*-3'	4128 Da.	4130 Da.

X designate ENA-T monomer. Small letters designate DNA monomers (a, g, c, t). Fitc = Fluorescein; EQL = Eclipse quencher; Dabcyl = Dabcyl quencher. MW = Molecular weight. Capital letters other than 'X' designate methyloxy LNA nucleotides.

REFERENCES AND NOTES

1. Helen C. Causton, Bing Ren, Sang Seok Koh, Christopher T. Harbison, Elenita Kanin, Ezra G. Jennings, Tong Ihn Lee, Heather L. True, Eric S. Lander, and Richard A. Young (2001). Remodelling of Yeast Genome Expression in Response to Environmental Changes. Mol. Biol. Cell 12:323-337 (2001).
2. Frank C. P. Holstege, Ezra G. Jennings, John J. Wyrick, Tong Ihn Lee, Christoph J. Hengartner, Michael R. Green, Todd R. Golub, Eric S. Lander, and Richard A. Young (1998). Dissecting the Regulatory Circuitry of a Eukaryotic Genome. Cell 1998 95: 717-728.
3. Simeonov, Anton and Theo T. Nikiforov, Single nucleotide polymorphism genotyping using short, fluorescently labelled locked nucleic acid (LNA) probes and fluorescence polarization detection, Nucleic Acid Research, 2002, Vol.30 No 17 e 91.

Variations, modifications, and other implementations of what is described herein will occur to those skilled in the art without departing from the spirit and scope of the invention as described and claimed herein and such variations, modifications, and implementations are encompassed within the scope of the invention.

The references, patents, patent applications, and international applications disclosed above are incorporated by reference herein in their entireties.

CLAIMS

1. A library of oligonucleotide probes wherein each probe in the library consists of a recognition sequence tag and a detection moiety wherein at least one monomer in each oligonucleotide probe is a modified monomer analogue, increasing the binding affinity for the complementary target sequence relative to the corresponding unmodified oligodeoxyribonucleotide, such that the library probes have sufficient stability for sequence-specific binding and detection of a substantial fraction of a target nucleic acid in any given target population and wherein the number of different recognition sequences comprises less than 10% of all possible sequence tags of a given length(s).

2. A library of oligonucleotide probes according to claim 1, wherein the recognition sequence tag segment of the probes in the library have been modified in at least one of the following ways:
i) substitution with at least one non-naturally occurring nucleotide
ii) substitution with at least one chemical moiety to increase the stability of the probe.

3. A library of oligonucleotide probes according to claim 1 or 2 wherein the recognition sequence tag has a length of 6 to 12 nucleotides.

4. A library of oligonucleotide probes according to claim 3, wherein the recognition sequence tag has a length of 8 or 9 nucleotides.

5. A library of oligonucleotide probes according to claim 4, wherein the recognition sequence tags are substituted with LNA nucleotides.

6. A library of oligonucleotide probes according to any one of the preceding claims, wherein more than 90% of the oligonucleotide probes can bind and detect at least two target sequences in a nucleic acid population.

7. A library according to claim 6, wherein the recognition sequence tag is complementary to at least two target sequences in the nucleic acid population.

8. A library of oligonucleotide probes of 8 and 9 nucleotides in length comprising a mixture of subsets of oligonucleotide probes defined in any one of claims 1-7.

9. A library of oligonucleotide probes of any one of the preceding claims, wherein the number of different target sequences in a nucleic acid population is at least 100.

10. A library of oligonucleotide probes according to any one of the preceding claims, wherein at least one nucleotide in each oligonucleotide probe is substituted with a non-naturally occurring nucleotide analogue, a deoxyribose or ribose analogue, or an internucleotide linkage other than a phosphodiester linkage.

11. A library of oligonucleotide probes according to any one of the preceding claims, wherein the detection moiety is a covalently or non-covalently bound minor groove binder or an intercalator selected from the group comprising asymmetric cyanine dyes, DAPI, SYBR Green I, SYBR Green II, SYBR Gold, PicoGreen, thiazole orange, Hoechst 33342, Ethidium Bromide, 1-O-(1-pyrenylmethyl)glycerol, and Hoechst 33258.

12. The library oligonucleotide probes according to claim 10 or 11, wherein the internucleotide linkage other than phosphodiester linkage is a non-phosphate internucleotide linkage.

13. The library of oligonucleotide probes according to claim 12, wherein the internucleotide linkage is selected from the group consisting of alkyl phosphonate, phosphoramidite, alkylphosphotriester, phosphorothioate, and phosphorodithioate linkages.

14. The library of oligonucleotide probes according to any one of the preceding claims, wherein said oligonucleotide probes contain non-naturally occurring nucleotides, such as 2'-O-methyl, diamine purine, 2-thio uracil, 5-nitroindole, universal or degenerate bases, intercalating nucleic acids or minor-groove-binders, to enhance their binding to a complementary nucleic acid sequence.

15. The library of oligonucleotide probes according to any one of the preceding claims, wherein said different recognition sequences comprise less than 1% of all possible oligonucleotides of a given length.

16. The library of oligonucleotide probes according to any one of the preceding claims, wherein each probe can be detected using a dual label by the molecular beacon assay principle.

17. The library of oligonucleotide probes according to any one of claims 1-15, wherein each probe can be detected using a dual label by the 5' nuclease assay principle.

18. The library according to any one of the preceding claims, wherein each probe contains a single detection moiety that can be detected by the molecular beacon assay principle.

19. The library of oligonucleotide probes according to any one of the preceding claims, wherein the target nucleic acid population is an mRNA sample, a cDNA sample or a genomic DNA sample.

20. The library of oligonucleotide probes according to claim 19, wherein said target mRNA or target cDNA population originates from the transcriptomes of human, mouse or rat.

21. The library of oligonucleotide probes according to any one of the preceding claims, wherein said probe target sequences occur at least once within more than 4% of different target nucleic acids in a target nucleic acid population.

22. The library of oligonucleotide probes according to any one of the preceding claims, wherein self-complementary probe sequences have been omitted from the said library.

23. The library of oligonucleotide probes according to claim 22, wherein said self-complementary sequences have been de-selected.

24. The library of oligonucleotide probes according to claim 22, wherein said self-complementary sequences have been eliminated by sequence-specific modifications, such as non-standard nucleotides, nucleotides with SBC nucleobases, 2'-O-methyl, diamine purine, 2-thio uracil, universal or degenerate bases or minor-groove-binders.

25. The library of oligonucleotide probes according to any one of the preceding claims, wherein the melting temperature (T_m) of each probe is adjusted to be suitable for PCR-based assays by substitution with non-occurring modifications, such as LNA, optionally modified with SBC nucleobases, 2'-O-methyl, diamine purine, 2-thio uracil, 5-nitroindole, universal or degenerate bases, intercalating nucleic acids or minor-groove-binders, to enhance their binding to a complementary nucleic acid sequence.

26. The library of oligonucleotide probes according to any one of the preceding claims, wherein the melting temperature (T_m) of each probe is at least 50°C.

27. The library of oligonucleotide probes according to any one of the preceding claims, wherein each probe has a DNA nucleotide at the 5'-end.

28. The library of oligonucleotide probes according to any one of the preceding claims, wherein each probe contains a fluorophore-quencher pair for detection.

29. The library of oligonucleotide probes according to any one of the preceding claims, wherein each probe can be detected by the molecular beacon principle.

30. The library of oligonucleotide probes according to any one of the preceding claims, wherein each probe is attached to an intercalating fluorophore or minor groove binder, which upon binding to a double-stranded DNA or DNA-RNA hetero-duplex target alter the fluorescence.

31. The library of oligonucleotide probes according to any one of the preceding claims, wherein the target population is the human transcriptome.

32. The library of oligonucleotide probes according to any one of the preceding claims, wherein each oligonucleotide probe detects the largest possible number of different target nucleic acids resulting in maximum coverage for a given target nucleic acid population by the said library.

33. The library of oligonucleotide probes according to any one of the preceding claims, wherein the oligonucleotide probes are selected to have as many target sequences or binding sites as possible within the target population of nucleic acids in order to obtain a maximum degree of detection.

34. The library of oligonucleotide probes according to any one of the preceding claims, wherein the oligonucleotide probes are selected to have at least one target sequence in as many target nucleic acids as possible within the target population of nucleic acids in order to obtain a maximum degree of detection.

35. The library of oligonucleotide probes in TABLE 1 or TABLE 1a or Fig. 13 or Fig. 14 capable of detecting the complementary sequences in any given nucleic acid population.

36. The library according to any one of the preceding claims, which comprises probes each having a recognition element listed in TABLE 1 or TABLE 1a in the specification and/or which comprises probes each having a recognition element complementary to the recognition elements listed in said TABLE 1.

37. An oligonucleotide probe useful in a library according to any one of the preceding claims, said probe being selected from probes complementary to or identical with the sequences set forth in Table 1, Fig. 13, or Fig 14.

38. An oligonucleotide probe according to claims 37, which has an exact nucleotide sequence selected from table 1.

39. A method of selecting oligonucleotide sequences useful in the library according to any one of the preceding claims, comprising
a) providing a first list of all possible oligonucleotides of a predefined number of nucleotides, N, said oligonucleotides having a melting temperature, T_m, of at least 50°C,
b) providing a second list of target nucleic acid sequences,
c) identifying and storing for each member of said first list, the number of members from said second list, which include a sequence complementary to said each member,
d) selecting a member of said first list, which in the identification in step c matches the maximum number, identified in step c, of members from said second list,
e) adding the member selected in step d to a third list consisting of the selected oligonucleotides useful in the library according to any one of the preceding claims,
f) subtracting the member selected in step d from said first list to provide a revised first list,
m) repeating steps d through f until said third list consists of members which together will be contemplary to at least 30% of the members on the list of target nucleic acid sequences from step b.

40. The method according to claim 39 , wherein T_m is at least 60°.

41. The method according to any claim 39 or 40, wherein the first list of oligonucleotides only includes oligonucleotides incapable of self-hybridization.

42. The method according to any one of claims 39-41, which after step f and before step m comprises the following steps:
g) subtracting all members from said second list which include a sequence complementary to the member selected in step d to obtain a revised second list,
h) identifying and storing for each member of said revised first list, the number of members from said revised second list, which include a sequence complementary to said each member,
i) selecting a member of said first list, which in the identification in step h matches the maximum number, identified in step h, of members from said second list, or selecting a member of said first list provides the maximum number obtained by multiplying the number identified in step h with the number identified in step c,
j) adding the member selected in step i to said third list,
k) subtracting the member selected in step i from said revised first list, and
l) subtracting all members from said revised second list which include a sequence or complementary to the member selected in step i.

43. The method according to any one of claims 39-42, wherein repetition in step m is continued until said third list consists of members which toghether will be contemplary to at least 85% of the members on the list of target nucleic acid sequences from step b.

44. The method according to any one of claims 39-43, wherein, after selection of the first member of said third list, the selection in step d after step c is preceded by identification of those members of said first list which hybridizes to more than a selected percentage of the maximum number of members from said second list so that only those members so identified are subjected to the selection in step d.

45. The method according to claim 44, wherein the selected percentage is 80%.

46. The method according to any one of claims 39-45, wherein N is an integer selected from 6, 7, 8, 9, 10, 11, and 12.

47. The method according to claim 46, wherein N is 8 or 9.

48. The method according to any one of claims 39-47, wherein said second list of step b comprises target nucleic acid sequences as defined in claim 19 or 20.

49. The method according to any one of claims 39-48, essentially performed as set forth in Fig. 2.

50. The method according to any one of claims 39-49, wherein said first, second and third lists are stored in the memory of a computer system, preferably in a database.

51. A computer program product providing instructions for implementing the method according to any one of claims 39-50, embedded in a computer-readable medium.

52. A system comprising a database of target sequences and an application program for executing the computer program of claim 51.

53. A method for identifying a specific means for detection of a target nucleic acid, the method comprising

A) inputting, into a computer system, data that uniquely identifies the nucleic acid sequence of said target nucleic acid, wherein said computer system comprises a database holding information of the composition of at least one library of nucleic acid probes according to any one of claims 1-36, and wherein the computer system further comprises a database of target

nucleic acid sequences for each probe of said at least one library and/or further comprises means for acquiring and comparing nucleic acid sequence data,

B) identifying, in the computer system, a probe from the at least one library, wherein the sequence of the probe exists in the target nucleic acid sequence or a sequence complementary to the target nucleic acid sequence,

C) identifying, in the computer system, primer that will amplify the target nucleic acid sequence, and

D) providing, as identification of the specific means for detection, an output that points out the probe identified in step B and the sequences of the primers identified in step C.

54. The method according to claim 53, wherein step A also comprises inputting, into the computer system, data that identifies the at least one library of nucleic acids from which it is desired to select a member for use in the specific means for detection.

55. The method according to claim 54, wherein the data that identifies the composition of the at least one library is a product code.

56. The method according to any one of claims 53-55, wherein inputting in step A is performed via an internet web interface.

57. The method according to any one of claims 53-55, wherein the primers identified in step C are chosen so as to minimize the chance of amplifying genomic nucleic acids in a PCR reaction.

58. The method according to claim 57, wherein at least one of the primers is selected so as to include a nucleotide sequence which in genomic DNA is interrupted by an intron.

59. The method according to any one of claims 53-58, wherein the primers selected in step C are chosen so as to minimize length of amplicons obtained from PCR performed on the target nucleic acid sequence.

60. The method according to any one of claims 53-59, wherein the primers selected in step C are chosen so as to optimize the GC content for performing PCR.

61. A computer program product providing instructions for implementing the method according to any one of claims 53-60 embedded in a computer-readable medium.

62. A system comprising a database of nucleic acid probes as defined in any one of claims 1-36 and an application program for executing the computer program of claim 61.

63. A method for profiling a plurality of target sequences comprising contacting a sample of target sequences with a library according to any one of claims 1-36 and detecting, characterizing or quantifying the probe sequences which bind to the target sequences.

64. The method according to claim 63, providing detection of a nucleic acid sequence which is present in less than 10% of the plurality of sequences which are bound by the multi-probe sequences.

65. The method according to claim 64, wherein the target mRNA sequences or cDNA sequences comprise a transcriptome.

66. The method according to claim 65, wherein the transcriptome is a human transcriptome.

67. The method according to any one of claims 63-66, wherein the library of probes are covalently coupled to a solid support.

68. The method according to claim 67, wherein the solid support comprises a microtiter plate and each well of the microtiter plate comprises a different library probe.

69. The method according to any one of claims 63-68, wherein the step of detecting is performed by amplifying a target nucleic acid sequence containing a recognition sequence complementary to a library probe.

70. The method of claim 69, wherein target nucleic acid amplification is carried out by using a pair of oligonucleoltide primers flanking the recognition sequence complementary to a library probe.

71. The method of claim 63-70, wherein the presence or expression level of one or more target nucleic acid sequences is correlated with a species' phenotype.

72. The method of claim 71, wherein the phenotype is a disease.

73. A method of analysing a mixture of nucleic acids using a library according to any one of claims 1-36 comprising the steps of

(a) contacting a target oligonucleotide with a library of labelled oligonucleotide probes, each of said oligonucleotide probes having a known sequence and being attached to a solid support at a known position, to hybridize said target oligonucleotide to at least one member of said library of probes, thereby forming a hybridized library;
(b) contacting said hybridized library with a nuclease capable of cleaving double-stranded oligonucleotides to release from said hybridized library a portion of said labelled oligonucleotide probes or fragments thereof; and
(c) identifying said positions of said hybridized library from which labelled probes or fragments thereof have been removed, to determine the sequence of said unlabelled target oligonucleotide.

74. A method of analysing a mixture of nucleic acids using a library of any one of claims 1-36 comprising the steps of
(a) contacting a target oligonucleotide with a library of labelled oligonucleotide probes, each of said oligonucleotide probes having a known sequence and being attached to a solid support at a known position, to hybridize said target oligonucleotide to at least one member of said library of probes, thereby forming a hybridized library;
(b) identifying said positions of said hybridized library at which labelled probes or fragments thereof have hybridized, to determine the sequence of said target oligonucleotide; and
(c) identifying said positions of said hybridized library from which labelled probes or fragments thereof have been removed, to determine the sequence of said unlabelled target oligonucleotide.

75. A method for quantitatively or qualitatively determining the presence of a target nucleic acid in a sample, the method comprising
i) identifying, by means of the method according to any one of claims 53-60, a specific means for detection of the target nucleic acid, where the specific means for detection comprises an oligonucleotide probe and a set of primers,
ii) obtaining the primers and the oligonucleotide probe identified in step i),
iii) subjecting the sample to a molecular amplification procedure in the presence of the primers and the oligonucleotide probe from step ii), and
iv) determining the presence of the target nucleic acid based on the outcome of step iii).

76. The method according to claim 75, wherein the primers obtained in step ii) are obtained by synthesis.

77. The method according to claim 75 or 76 or, wherein the oligonucleotide probe is obtained from a library according to any one of claims 1-36.

78. The method according to any one of claims 75-77, wherein the procedure in step iii) is a PCR or a NASBA procedure.

79. The method according to claim 78, wherein the PCR procedure is a qPCR.

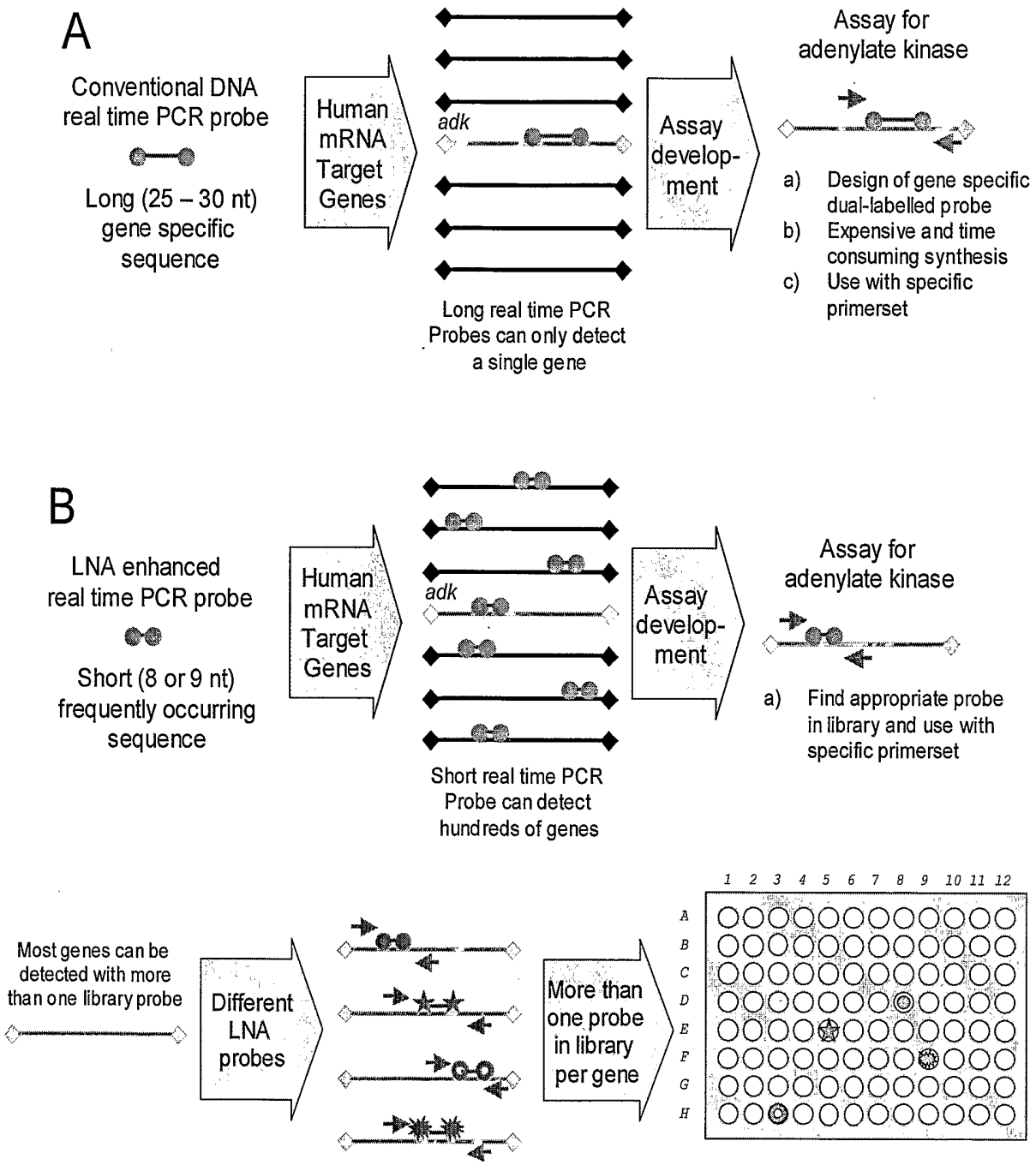

A
Conventional DNA real time PCR probe
Long (25 – 30 nt) gene specific sequence
Human mRNA Target Genes
adk
Long real time PCR Probes can only detect a single gene
Assay develop-ment
Assay for adenylate kinase
a) Design of gene specific dual-labelled probe
b) Expensive and time consuming synthesis
c) Use with specific primerset
B
LNA enhanced real time PCR probe
Short (8 or 9 nt) frequently occurring sequence
Human mRNA Target Genes
adk
Short real time PCR Probe can detect hundreds of genes
Assay develop-ment
Assay for adenylate kinase
a) Find appropriate probe in library and use with specific primerset
Most genes can be detected with more than one library probe
Different LNA probes
More than one probe in library per gene
1 2 3 4 5 6 7 8 9 10 11 12
A
B
C
D
E
F
G
H

Fig. 1

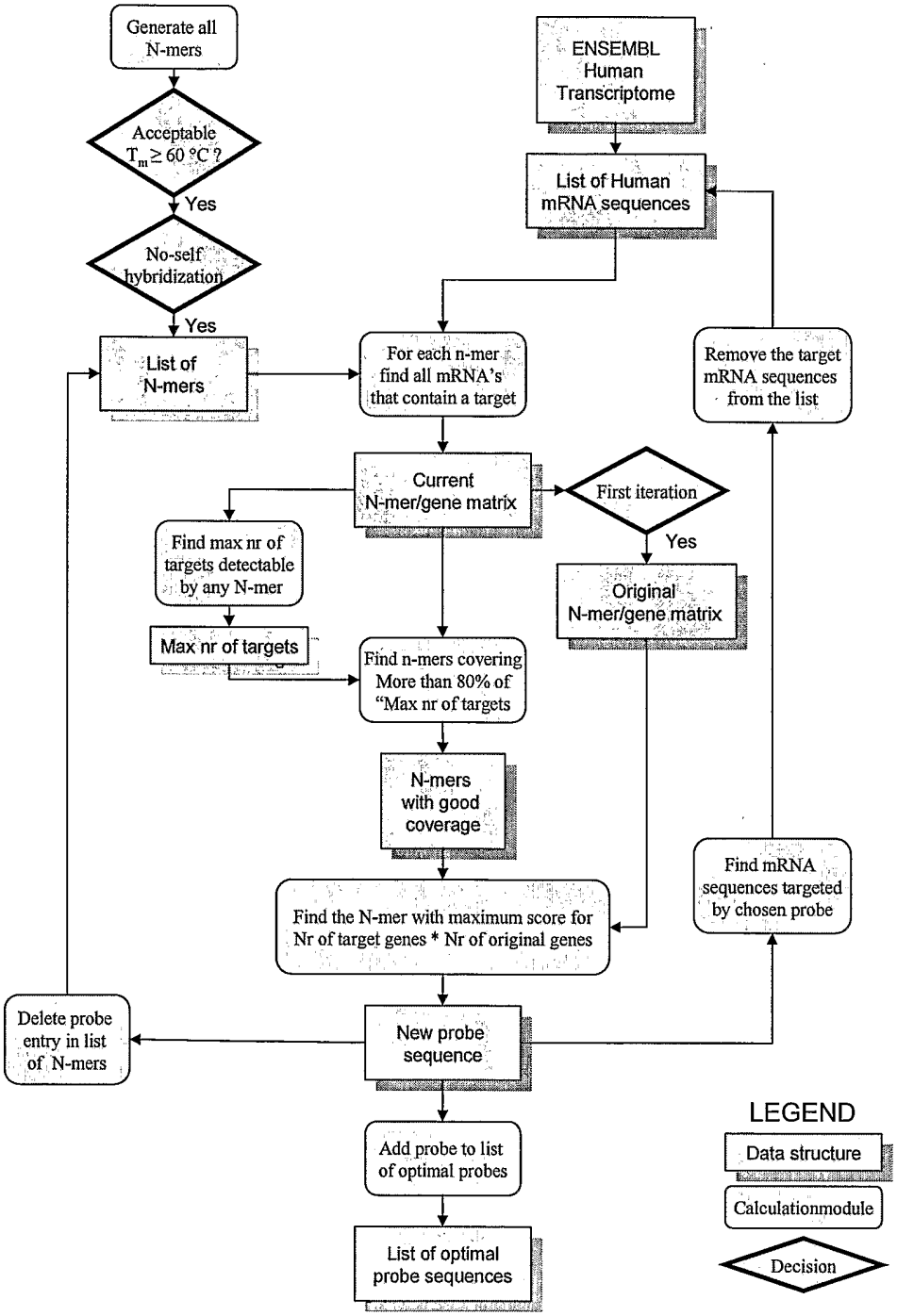

Generate all N-mers
Acceptable $T_m \geq 60$ °C ?
Yes
No-self hybridization
Yes
List of N-mers
ENSEMBL Human Transcriptome
List of Human mRNA sequences
For each n-mer find all mRNA's that contain a target
Remove the target mRNA sequences from the list
Current N-mer/gene matrix
First iteration
Yes
Original N-mer/gene matrix
Find max nr of targets detectable by any N-mer
Max nr of targets
Find n-mers covering More than 80% of "Max nr of targets
N-mers with good coverage
Find mRNA sequences targeted by chosen probe
Find the N-mer with maximum score for Nr of target genes * Nr of original genes
Delete probe entry in list of N-mers
New probe sequence
Add probe to list of optimal probes
List of optimal probe sequences
LEGEND
Data structure
Calculationmodule
Decision

Fig. 2

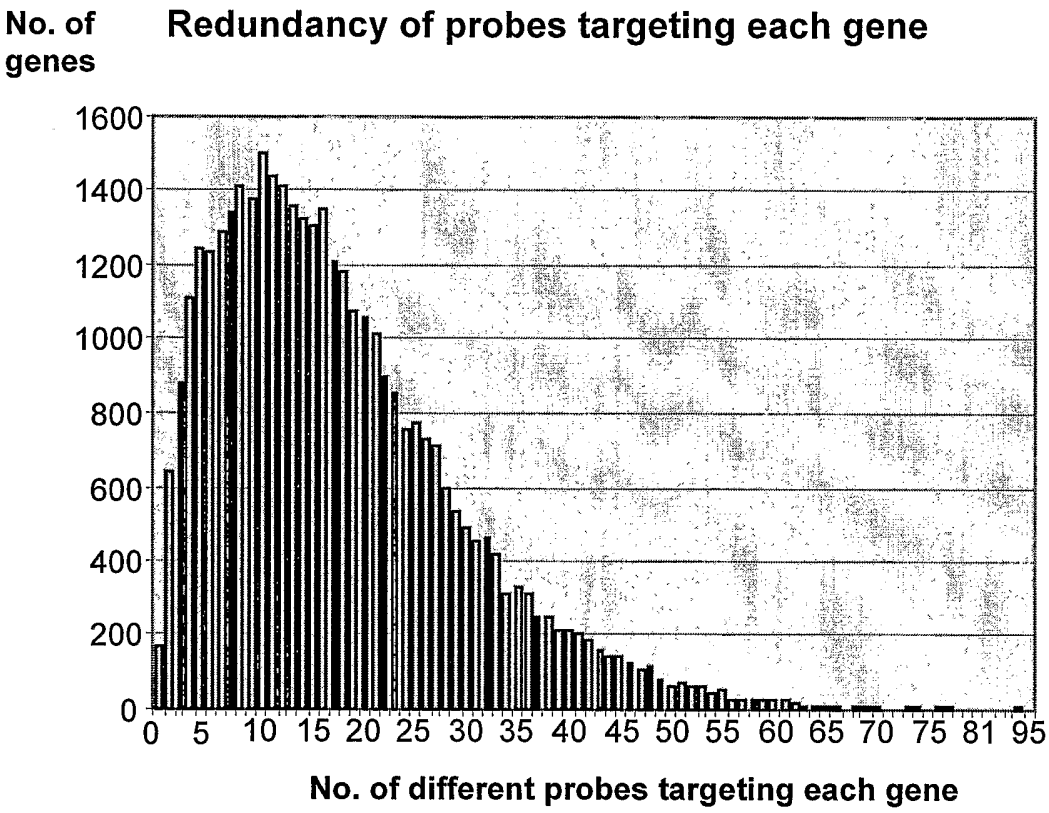

No. of genes
Redundancy of probes targeting each gene
1600
1400
1200
1000
800
600
400
200
0
0 5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 81 95
No. of different probes targeting each gene

Fig. 3

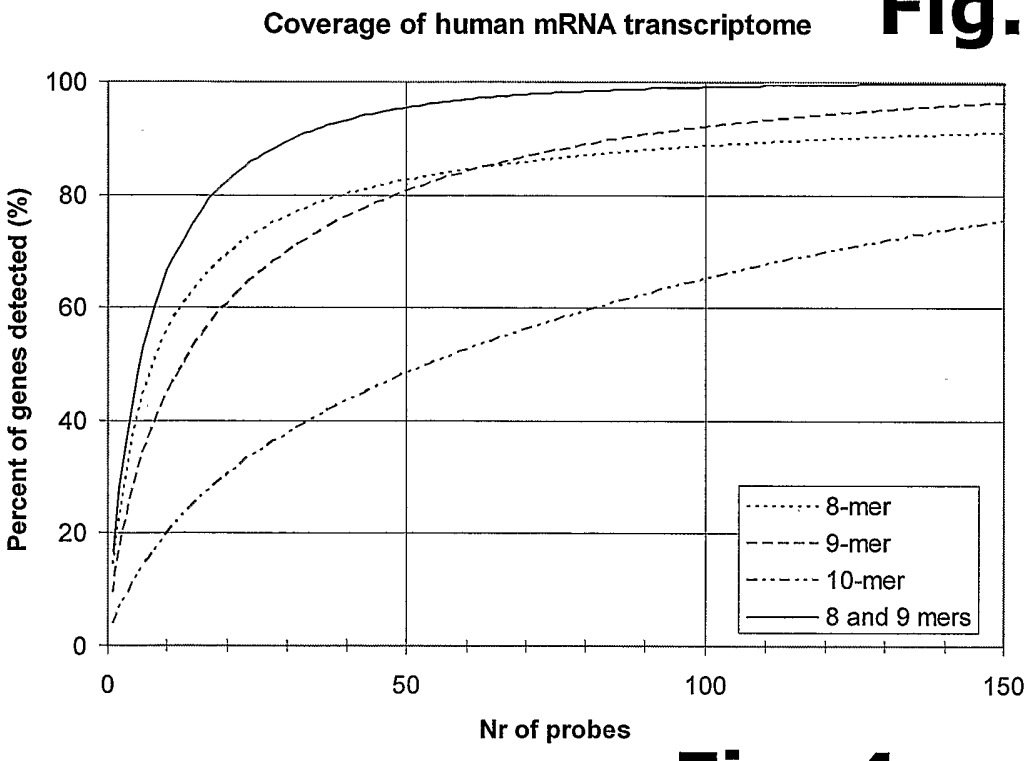

Coverage of human mRNA transcriptome
Percent of genes detected (%)
100
80
60
40
20
0
0
50
100
150
Nr of probes
8-mer
9-mer
10-mer
8 and 9 mers

Fig. 4

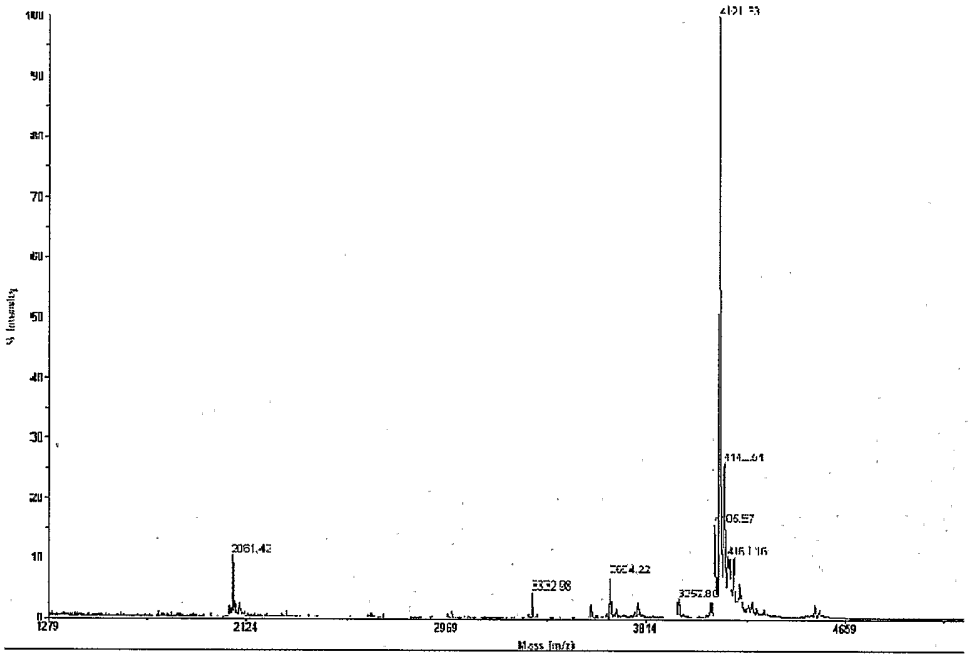

100
90
80
70
60
50
40
30
20
10
0
% Intensity
2061.42
3332.98
3664.22
4101.53
4114.61
4136.57
4161.16
1279
2124
2969
3814
4659
Mass [m/z]

Fig. 5

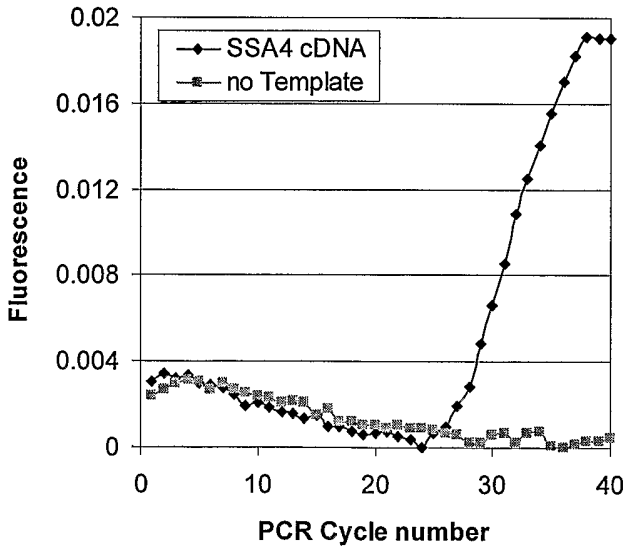

0.02
0.016
0.012
0.008
0.004
0
Fluorescence
SSA4 cDNA
no Template
0
10
20
30
40
PCR Cycle number

Fig. 8

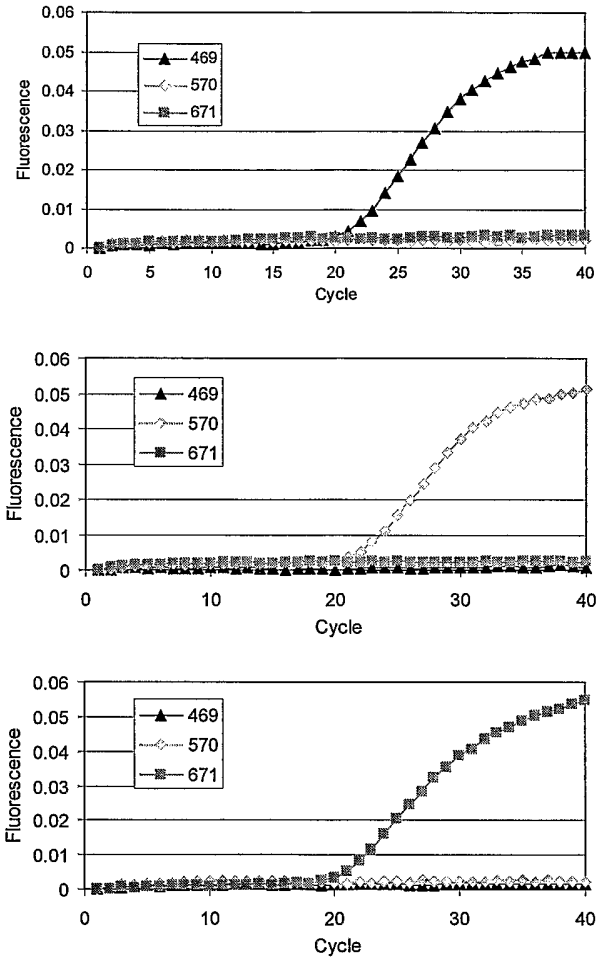

0.06
0.05
0.04
0.03
0.02
0.01
0
Fluorescence
469
570
671
0 5 10 15 20 25 30 35 40
Cycle
0.06
0.05
0.04
0.03
0.02
0.01
0
Fluorescence
469
570
671
0 10 20 30 40
Cycle
0.06
0.05
0.04
0.03
0.02
0.01
0
Fluorescence
469
570
671
0 10 20 30 40
Cycle

Fig. 6

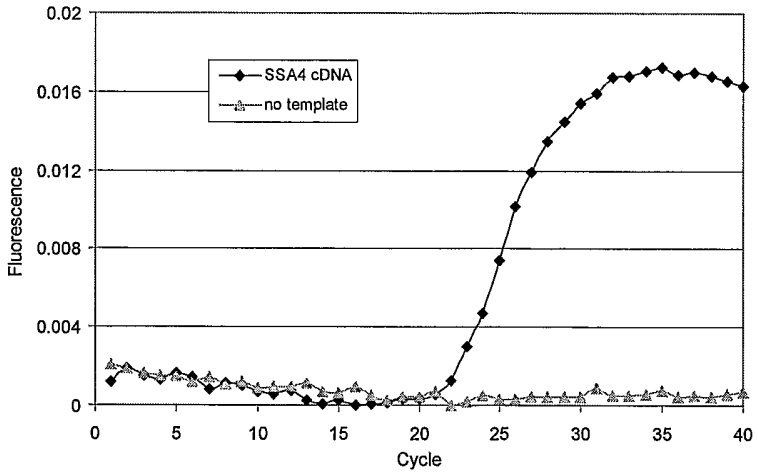

0.02
0.016
0.012
0.008
0.004
0
Fluorescence
SSA4 cDNA
no template
0
5
10
15
20
25
30
35
40
Cycle

Fig. 7A

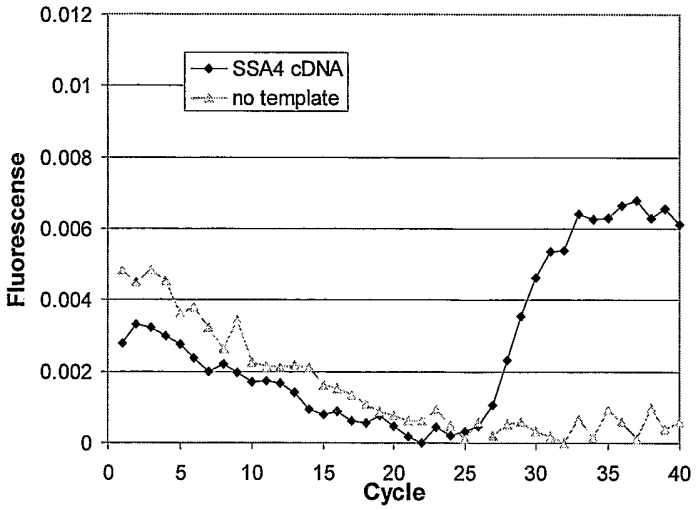

0.012
0.01
0.008
0.006
0.004
0.002
0
Fluorescense
SSA4 cDNA
no template
0
5
10
15
20
25
30
35
40
Cycle

Fig. 7B

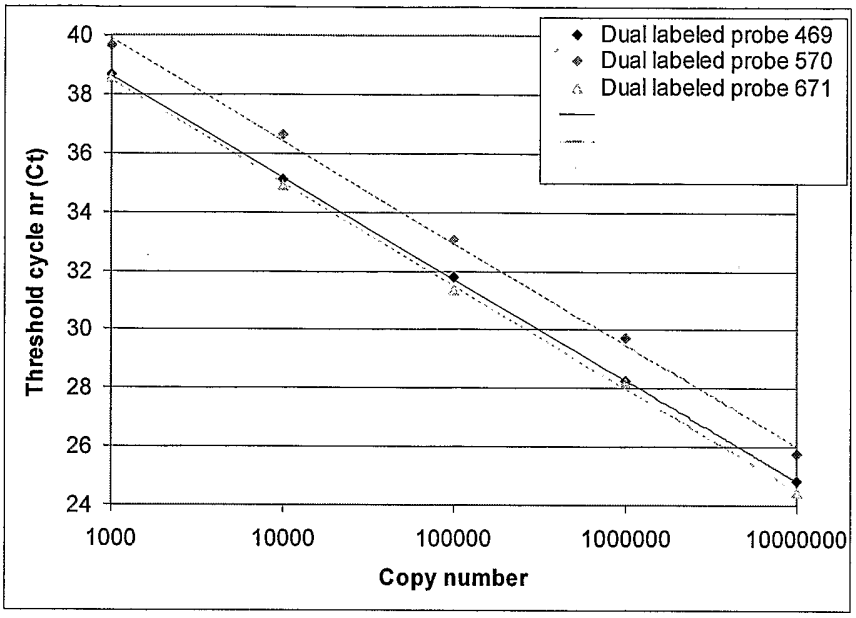

Dual labeled probe 469
Dual labeled probe 570
Dual labeled probe 671
Threshold cycle nr (Ct)
40
38
36
34
32
30
28
26
24
1000
10000
100000
1000000
10000000
Copy number

Fig. 9

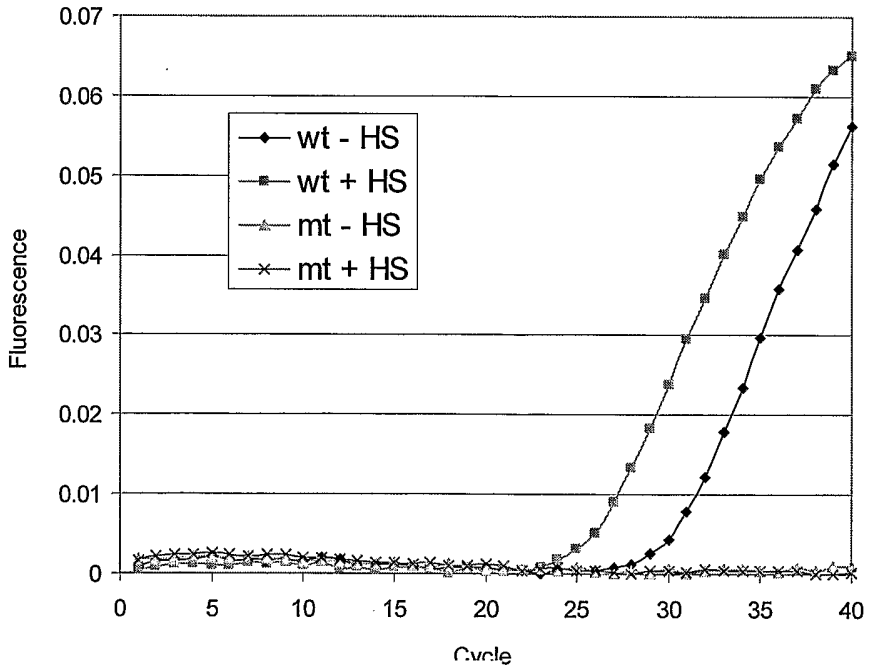

wt - HS
wt + HS
mt - HS
mt + HS
Fluorescence
0.07
0.06
0.05
0.04
0.03
0.02
0.01
0
0
5
10
15
20
25
30
35
40
Cycle

Fig. 10

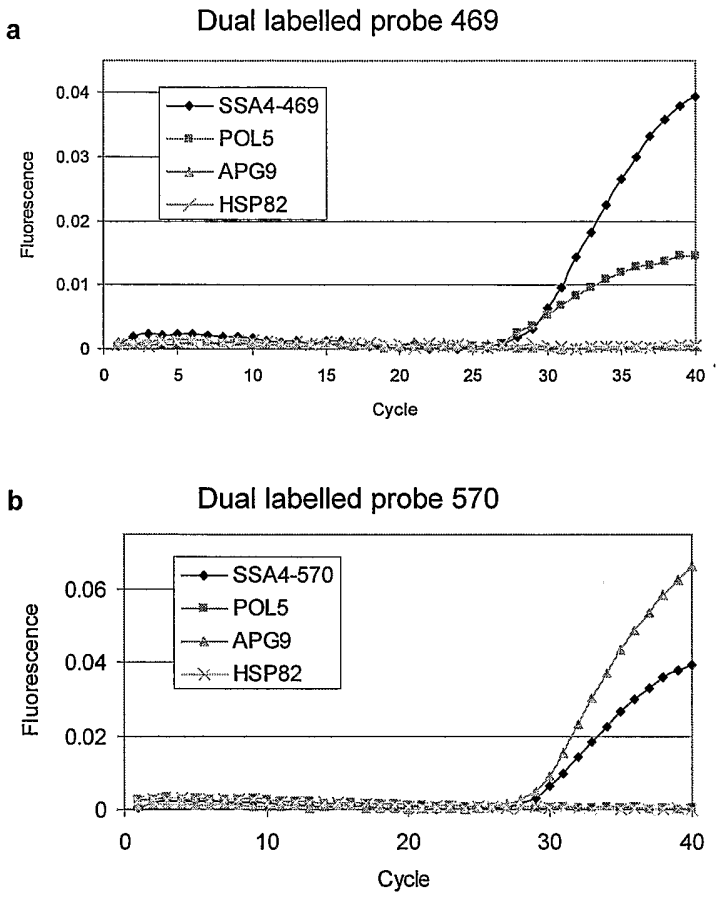

a
Dual labelled probe 469
SSA4-469
POL5
APG9
HSP82
Fluorescence
Cycle
b
Dual labelled probe 570
SSA4-570
POL5
APG9
HSP82
Fluorescence
Cycle
c
Dual labelled probe 671
SSA4-671
POL5
APG9
HSP82
Fluorescence
Cycle

Fig. 11

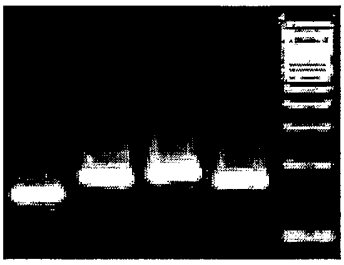


Fig. 12

1	ctcctcct	Lib_hum_A_005	42	cttctccc	Lib_hum_B_127
2	ctggagga	Lib_hum_A_007	43	cctcagcc	Lib_hum_C_202
3	aggagctg	Lib_hum_B_101	44	ctccttcc	Lib_hum_C_246
4	cagcctgg	Lib_hum_A_008	45	cagcaggc	Lib_hum_C_203
5	cagcagcc	Lib_hum_A_009	46	ctgcctct	Lib_hum_C_204
6	tgctggag	Lib_hum_A_010B	47	ctccacct	Lib_hum_C_205
7	agctggag	Lib_hum_A_019	48	ctcctccc	Lib_hum_C_206
8	ctgctgcc	Lib_hum_A_020	49	cttcccca	Lib_hum_C_207
9	aggagcag	Lib_hum_A_011B	50	cttcagcc	Lib_hum_C_208
10	ccaggagg	Lib_hum_A_016	51	ctctgcca	Lib_hum_C_209
11	tcctgctg	Lib_hum_B_102	52	ctgggaga	Lib_hum_C_247
12	cttcctcc	Lib_hum_A_015	53	cttctgcc	Lib_hum_C_210
13	ccgccgcc	Lib_hum_A_004	54	cagcaggt	Lib_hum_C_211
14	cctggagc	Lib_hum_B_103	55	tctggagc	Lib_hum_C_214
15	cagcctcc	Lib_hum_A_017	56	tcctgctc	Lib_hum_C_248
16	tggctgtg	Lib_hum_A_021	57	ctggggcc	Lib_hum_C_220
17	cctggaga	Lib_hum_A_022	58	ctcctgcc	Lib_hum_C_219
18	ccagccag	Lib_hum_B_105	59	ctgggcaa	Lib_hum_C_223
19	ccagggcc	Lib_hum_A_023	60	ctggggct	Lib_hum_C_226
20	cccagcag	Lib_hum_B_106	61	tggtggcc	Lib_hum_C_225
21	ccaccacc	Lib_hum_A_025	62	ccagggca	Lib_hum_C_240
22	ctcctcca	Lib_hum_B_104	63	ctgctccc	Lib_hum_C_227
23	ttctcctg	Lib_hum_B_109	64	tgggcagc	Lib_hum_C_239
24	cagcccag	Lib_hum_B_110	65	ctccatcc	Lib_hum_C_222
25	ctggctgc	Lib_Hum_A_001	66	ctgcccca	Lib_hum_C_215
26	ctccacca	Lib_hum_B_112	67	ttcctggc	Lib_hum_C_244
27	cttcctgc	Lib_hum_B_111	68	atggctgc	Lib_hum_C_217
28	cttccagc	Lib_hum_B_114	69	tggtggaa	Lib_hum_C_231
29	ccacctcc	Lib_hum_B_121	70	tgctgtcc	Lib_hum_C_228
30	ttcctctg	Lib_hum_B_122	71	ccagccgc	Lib_hum_C_224
31	cccagccc	Lib_hum_B_116	72	catccagc	Lib_hum_C_216
32	tggtgatg	Lib_hum_B_124	73	tcctctcc	Lib_hum_C_229
33	tggctctg	Lib_hum_B_123	74	agctggga	Lib_hum_C_233
34	ctgccttc	Lib_hum_B_118	75	ctggtctc	Lib_hum_C_234
35	ctccagcc	Lib_hum_B_119	76	ttcccagt	Lib_hum_C_235
36	tgtggctg	Lib_hum_B_125	77	caggcagc	Lib_hum_C_250
37	cagaggag	Lib_hum_A_048B	78	tcctcagc	Lib_hum_C_245
38	cagctccc	Lib_hum_B_129	79	ctggctcc	Lib_hum_C_241
39	ctgcctcc	Lib_hum_B_128	80	tcctcttct	Lib_hum_C_236
40	tctgctgc	Lib_hum_C_242	81	tccagtgt	Lib_hum_C_237
41	ctgcttcc	Lib_hum_C_201	82	acagcctca	GAPDH_AQQ1
			83	cagccacc	Lib_hum_C_243

Fig. 13

84	cttctggc	Lib_hum_D_301
85	tccactgc	Lib_hum_D_302
86	ctctgcgt	Lib_hum_D_303
87	ctggaggc	Lib_hum_D_304
88	ctggaagc	Lib_hum_D_305
89	ctgccacc	Lib_hum_D_306
90	tccaggtc	Lib_hum_D_307
91	cagcatcc	Lib_hum_D_308
92	cagaggct	Lib_hum_D_309
93	ctgaagcc	Lib_hum_D_310
94	tgcagggc	Lib_hum_D_311
95	atggcagc	Lib_hum_D_312
96	catcctcc	Lib_hum_D_313
97	ctctgcct	Lib_hum_D_314
98	ccagtgcc	Lib_hum_D_315
99	cagtggca	Lib_hum_D_316
100	actgctgc	Lib_hum_D_317
101	cagcaccc	Lib_hum_D_318
102	atgatggc	Lib_hum_D_319
103	ccaggagc	Lib_hum_D_320

Fig. 14

14590	Lib_hum_A_con1_target	caccgttcggcatatccatatttcccacagccaccaccaggaaggcagccagcaggaggagcagcctcct cagagaagcagcctggagacttcctccagctccagggccgccgcctgctggagcagcagcaccagaaagag ggggaggtacggttggttgtacga
14778	Long mere 001-015-sense (Hum Lib A 001-014)	tggcggacgcacaccgcttaccctgctggaggaagctgaggaggagcagcctggagcagcagccagccag ctccgccgccaggaagccgactcacgggccacgcatta
14779	Long mere 016-026-antisense (Hum Lib A 016-026, 011, 048)	gggtgcgaccgtgagtcaatggtctccaggaggctgtcttctggtgctgctcctctgctgcctccagctt ctctggccctggtggtgctgggtaatgcgtggcccgtgagtc
15315	Long mere B1a-sense	attgactcacggtcgcaccaaactctgctgggctgcctggaaagctccaggagaacttccagccagctcct ccaccagcaggaagaataaccgtggaacgcggtcat
15316	Long mere B1b-antisense	atacccatccaaggcgtccctaaaggaggcagaggaagggagctgcctcccagccctttctcccagcaca gcagagcagagccacctccagccacatcaccaaaatgaccgcgttccacggtta
15317	Long mere B2a-sense	attgactcacggtcgcaccaaacctggaaggcagaggaactgcctcctccaccatcaccactgctgggct gggaagcttccagcacaggcaggaaataaccgtggaacgcggtcat
15318	Long mere B2b-antisense	atacccatccaaggcgtccctaaacttctcccagagccacctccagccagccacaccaggcagagcaggaa ggagctgcctggagcagctcccaggagaaaaatgaccgcgttccacggtta
15319	Long mere B3a-sense	attgactcacggtcgcaccaaattcctctgccttcctgctctgctgggagaaggaggtggtgatgtggct ggaaggaggcagctccaggagaaaataaccgtggaacgcggtcat
15320	Long mere B3b-antisense	atacccatccaaggcgtccctaaacttccaggcagctccctccagccagcaggactttcccagcccaggctc ctccaccagcacagcagagccaaaatgaccgcgttccacggtta
15430	Long mere E1 – sense	ttagggacgccttggatgggtatggctgaggcggctggctcctgcatcctcttctgcctctgctcccagc tgagccatgccctggcttccaccaattgccgacccaccgggata
15431	Long mere F1 – antisense	Attcgctacggcccaacacctttactccacctcctgccccactgggctgaagtccagt- gtctggagctgcttcccagtgggcagccatccagcaggccaccatatcccggtgggtcggcaat
15432	Long mere G1 – sense	taaggtgttgggccgtagcgaatcgctctgccact- gggcctggtctccatcctcctccctgggcaacctgctgtcctggcagt- ggggaagctgtgccaattgtcctccgccgggactcat
15433	Long mere E2 – sense	ttagggacgccttggatgggtatctctgccactggctccagatcctcttctgccccact- gccatgggcagctgggggcctcctccctccacctggcttcccccaattgccgacccaccgggata
15434	Long mere F2- antisense	attcgctacggcccaacacctttacctcagccccagctccatccagccgccaag- gactggtctcctgccctgggcaactgggaatggctgcttccaccatatcccggtgggtcggcaat
15435	Long mere G2 – sense	taaggtgttgggccgtagcgaatctgcctcttcagccgctctgctcccagctgagccatc- cagtgtgcaggagaggacagcaggtggcacagcaggccaccaattgtcctccgcccgggactcat

Fig. 15

qPCR for human genes made easy.

The design of an efficient and reliable qPCR assay for a human gene is a complex task. We here present the ProbeFinder (see www.probelibrary.com) a new web tool for fast and easy selection of ProbeLibrary™ probes and the design of primers for qPCR of human genes.
The ProbeFinder web server designs optimal qPCR probes and primers fast and reliably for any given human gene. Alternative solutions for genes with special requirements are presented on easy to use web pages.

ProbeFinder designs the optimal qPCR in three steps.

Determine intron positions

Noise from chromosomal DNA* is eliminated by selecting intron-spanning qPCR's. Introns are determined by a blast search against the human genome. Regions found on the DNA, but not in the transcript are considered to be introns.

* The intron prediction can be deselected for qPCR's that are free from chromosomal DNA.

Match ProbeLibrary™ to gene

Virtually all human transcripts* are covered by at least one of the 90 ProbeLibrary™ probes. The high coverage is made possible using the LNA™ technology of Exiqon. The matching probe is identified by ProbeFinder™ within seconds.

* A human transcript from RefSeq is on average covered by 17 ProbeLibrary™ probes and each probe is on average found in more than 7000 of the 38556 transcripts.

Design primers and select optimal qPCR assay

Primers are designed with Primer3. Finally the assays are ranked according to carefully selected rules ensuring the best possible qPCR assay. The rules favour intron spanning amplicons to remove false signals from DNA contamination, small amplicon size for reproducible and comparable assays and a GC content optimised for PCR.

Fig. 16

Listing of program modules

Program main.aap

```
CC = gcc -DLIB -O2

SOURCE =
        getcover.c
        dyp.c
        getopt.c
        getopt1.c
        getopt_init.c

LIBS   = -lm

TARGET = getcover$EXESUF
```

Fig. 17

Program dyp.h

```
/*AUTHOR
        Copyright Niels Tolstrup 2003
        Exiqon
*/

typedef struct sequences_t {
         int            nseq;
         int            maxnseq;
         char           **seqs;
        } sequences_type;


typedef struct score_rec_t {
         int            alph_len;
         int            gap_start;
         int            gap_cont;
         int            mismatch_cont;
         int            loop_score;
         int            match_cont_factor;
         int            match_threshold;
         int            min_strong_ident_score;
         int            min_ident_score;
         int            min_sim_score;
         char           *alph;
         int            *mat;
        } score_rec_type;


typedef struct param_t {
         char           *alph;
         int            *mathyp;
         int            depth;
         int            global;
         int            min_score;
         int            max_res;
         sequences_type *seqs;
         sequences_type *seqs2;
         score_rec_type *score_rec;
        } param_type;


typedef struct dynamic_str_t {
         int            len;
         int            maxlen;
         char           *s;
        } dynamic_str_type;
```

Fig. 17 (contd.)

```
param_type *init();

char *empty_str( int len);

char *make_nmer( long number, int n_mer, char *alph);

dynamic_str_type *new_dynamic_str_type();

dynamic_str_type *addchar( dynamic_str_type *seq, int
c);

long n_nmer( int n_mer, char *alph);

long reverse_dna( long n, int oligolen);

long comp_dna( long n, int oligolen);

long rev_comp_dna( long n, int oligolen);
```

Program getcover.c

```
/*

 #$Id: $
SYNOPSIS
      getcover -l 9 -p -f < h_sap.fasta > h_sap_l9.stat
        getcover -l 9 -i 1 -d 10 -t 60 -c -n -m -s <
h_sap_l9.stat > h_sap_l9.cover


DESCRIPTION
        Find a cover of n mers for human transcriptome
        Input: A fasta file
        Options: getcover --help

INSTALATION
      The program works udner solaris and linux.


AUTHOR
        Copyright Niels Tolstrup 2003
        Exiqor

*/
```

(contd.)

Fig. 17 (contd.)

```
#include <stdio.h>    /* fprintf, rand..*/
#include <stdlib.h>   /* calloc..       */
#include <stdarg.h>   /* va_list..      */
#include <string.h>   /* strlen..       */
#include <limits.h>   /* USHRT_MAX..    */
#include <math.h>     /* pow            */
#include <ctype.h>    /* tolower        */
#include <time.h>     /* clock          */
#include "dyp.h"      /* make_nmer..    */
#include "getopt.h"   /* getopt_long    */

/*********** GLOBAL VARIABLES ***************/

#ifdef NODYP
int             verbose = 0;
char*           program_name;
#else
extern int verbose;
extern char* program_name;
#endif


int             t_start;
int             t_lap1;
int             t_lap2;
int             t_run;
int             t_debug;
int             t_last_debug;
int             t_debug_interval = 5;

/*** these should not be global ***/

#define TIME_INIT  0
#define LAP_TIME   1
#define TOTAL_TIME 2
#define DEBUG_TIME 4


/************ STRUCTURES *********************/

typedef struct alph_t {
          char         *name;
          char         *alph;
          int          len;
        } alph_type;
```

Fig. 17 (contd.)

```
typedef struct lna_t {
          long          lna_id;
          char          self_score;
          char          target_score;
          char          tm;
        } lna_type;

typedef struct nmer_t {
          int           nmer_id;
          short         selected;
          char          ok;       /*
==0:undefined,<0:discard,>0:ok */
          lna_type      *lna;
          int           n_total_gene;
          int           n_gene;
          int           max_n_gene;
          int           *gene;
        } nmer_type;


typedef struct stat_t {
          int           n_nmer;
          int           oligolen;
          alph_type     *alph;
          int           max_n_nmer;
          char          compact;
          nmer_type     **nmer;
          int           n_gene;
          int           max_n_gene;
          int           *gene;
        } stat_type;

typedef struct conf_t {
          int           fasta_in;
          int           stat_in;
          int           oligolen;
          float         max_gene_hit_frac;
          int           max_gene_hit;
          int           target_minus_self;
          int           target_min_temp;
          int           complement_flag;
          int           max_select;
          int           no_5end_spike;
          int           max_product;
        } conf_type;
```

Fig. 17 (contd.)

```
195

      #ifdef NODYP
      void die( char *fmt, ...){
        va_list ap;
200     fprintf( stderr, "Uhoh: ");
        va_start( ap, fmt);
        vfprintf( stderr, fmt, ap);
        va_end( ap);
        fprintf( stderr, "\n");
205     exit( 1);
      }
      #endif


210   void print_debug( char *fmt, ...){
        va_list ap;
        if( verbose >= 3) {
          va_start( ap, fmt);
          vfprintf( stderr, fmt, ap);
215       va_end( ap);
          fprintf( stderr, "\n");
          fflush( stderr);
        }
      }
220

      void print_debug_interval( char *fmt, ...){
        va_list ap;
        if( (verbose >= 3) && run_time( DEBUG_TIME)) {
225       fprintf( stderr, "%8d ", t_run);
          va_start( ap, fmt);
          vfprintf( stderr, fmt, ap);
          va_end( ap);
          fprintf( stderr, "\n");
230       fflush( stderr);
        }
      }


235   void usage( char *fmt, ...){
        va_list ap;
        va_start( ap, fmt);
        vfprintf( stderr, fmt, ap);
        va_end( ap);
240     fprintf( stderr,
            "\nUsage: getcover -h,--help\n"
            "    -v,--verbose\n"
            "    -vv,--verbose --verbose\n"
```

Fig. 17 (contd.)

```
            "    -l,--oligo_len len\n"
            "    -i,--max_gene_hit_frac fraction\n"
            "    -d,--target_minus_self delta\n"
            "    -t,--target_min_temp temp\n"
            "    -c,--complement\n"
            "    -m,--max_product\n"
            "    -n,--no_5end_spike\n"
            "    -f,--fasta_in\n"
            "    -s,--stat_in\n"
            "    -p,--dump_stat\n"
            "\n"
            "DESCRIPTION\n"
            "     ");
        fprintf( stderr, "\n");
        exit( 1);
    }

    int run_time( int type){
        if( type == DEBUG_TIME) {
            t_debug = clock()/CLOCKS_PER_SEC;
            if( t_debug - t_last_debug >= t_debug_interval ) {
                t_run = t_debug - t_start;
                t_last_debug = t_debug;
            }
            else
                t_run = 0;
        }
        else if( type == TIME_INIT) {
            t_start       = clock()/CLOCKS_PER_SEC;
            t_lap1        = t_start;
            t_last_debug  = t_start;
            t_run         = 0;
        }
        else if( type == LAP_TIME) {
            t_lap2  = clock()/CLOCKS_PER_SEC;
            t_run   = t_lap2 - t_lap1;
            t_lap1  = t_lap2;
        }
        else if( type == TOTAL_TIME) {
            t_lap1  = clock()/CLOCKS_PER_SEC;
            t_run   = t_lap1 - t_start;
        }
        return t_run;
    }

    #ifdef NODYP
    void *salloc( int nobj, int size){
```

Fig. 17 (contd.)

```
  void *mem;
  /*mem =       calloc( nobj, size);*/
  mem = malloc( nobj * size);
  /*printf( "calloc: allocating %d objects of size %d
bytes\n", nobj, size);*/
  if( mem == NULL){
    die( "Could not allocate %d x %d bytes\n", nobj,
size);
  }
  return mem;
}
#endif


void free_nmer_type( nmer_type *nmer) {
  if( nmer != NULL ) {
    free( nmer->gene);
    free( nmer->lna);
  }
  free( nmer);
}


void free_stat( stat_type *stat) {
  int       i;

  for( i=0; i<stat->n_nmer; i++)
    free_nmer_type( stat->nmer[i]);

  free_alph_type( stat->alph);
  free( stat->nmer);
  free( stat);
}


/* set, union, intersection, complement would be nice
to have */

  /* the lists must be sorted for this algorith! */

void set_union( int *list1, int n_list1, int *list2,
int n_list2,
                int **reslist, int *n_reslist){
  int i, j, k, n;
  int n1, n2;
  int *tmplist;

  tmplist       = (int *) salloc( n_list1 + n_list2,
sizeof( int));
```

Fig. 17 (contd.)

```
        j=0;
        k=0;
345     n=0;
        while( j<n_list1 && k<n_list2 ) {
          n1  = list1[j];
          n2  = list2[k];
          if( n1 < n2 ) {
350         tmplist[ n] = n1;
            n++;
            j++;
          }
          else if( n2 < n1 ) {
355         tmplist[ n] = n2;
            n++;
            k++;
          }
          else {
360         tmplist[ n] = n1;
            n++;
            j++;
            k++;
          }
365     }

        for( ; j<n_list1; j++) {
          n1  = list1[j];
          tmplist[ n] = n1;
370       n++;
        }

        for( ; k<n_list2; k++) {
          n2  = list2[k];
375       tmplist[ n] = n2;
          n++;
        }

        *n_reslist  = n;
380     *reslist    = (int *) salloc( *n_reslist, sizeof(
      int));

        for( k=0; k<n; k++) {
           (*reslist)[k] = tmplist[ k];
385     }
        free( tmplist);
      }


390   stat_type *compact_stat( stat_type *stat){
```

Fig. 17 (contd.)

```
    int i, j;

    for( i=0; i<stat->max_n_nmer && stat->nmer[i] != NULL
              && stat->nmer[i]->n_gene > 0; i++)
395  ;

    j=i;
    for( ; i<stat->max_n_nmer; i++) {
      if(stat->nmer[i] != NULL) {
400     if( stat->nmer[i]->n_gene > 0) {
          stat->nmer[j] = stat->nmer[i];
          stat->nmer[i] = NULL;

          j++;
405     }
        else {
          free_nmer_type( stat->nmer[i]);
          stat->nmer[i] = NULL;
        }
410   }
    }
    stat->n_nmer  = j;
    stat->compact = 1;
  }
415

  stat_type *comp_stat( stat_type *stat){
    int       i, j, comp_i, comp_j;
    char      *seq, *compseq;
420 char      alph[10] = "acgt";
    int       mask = (n_nmer( stat->oligolen, alph) - 1);
    int       *tmpgene;
    int       ngene;
    int       comp_nmer_id;
425 nmer_type *the_nmer, *the_comp_nmer;


  if( stat->compact )
    die("Internal error, comp_stat does not work for com-
430 pact stat");


    for( i=0; i<stat->max_n_nmer; i++) {
      the_nmer = stat->nmer[i];
435   if( the_nmer != NULL ) {
        comp_nmer_id  = rev_comp_dna( the_nmer->nmer_id,
  stat->oligolen);
        the_comp_nmer = NULL;
```

Fig. 17 (contd.)

```
440         comp_j = comp_nmer_id;
            the_comp_nmer = stat->nmer[ comp_j];
            /*
            for( j=0; j<stat->n_nmer && the_comp_nmer ==
    NULL; j++) {
445           if( stat->nmer[j] != NULL &&
                  stat->nmer[j]->nmer_id == comp_nmer_id) {
              comp_j        = j;
              the_comp_nmer = stat->nmer[ comp_j];
            }
450         }
            */

            if( the_comp_nmer != NULL && comp_nmer_id !=
    the_nmer->nmer_id) {
455           set_union( the_nmer->gene, the_nmer->n_gene,
                         the_comp_nmer->gene, the_comp_nmer-
    >n_gene,
                         &tmpgene, &ngene);
              if( the_nmer->nmer_id < the_comp_nmer->nmer_id
460   ) {
                the_nmer->n_gene       = ngene;
                the_nmer->n_total_gene = ngene;
                free( the_nmer->gene);
                the_nmer->gene   = tmpgene;
465             free_nmer_type( the_comp_nmer);
                stat->nmer[ comp_j] = NULL;
              }
              else if( the_nmer->nmer_id > the_comp_nmer-
    >nmer_id ) {
470             the_comp_nmer->n_gene       = ngene;
                the_comp_nmer->n_total_gene = ngene;
                free( the_comp_nmer->gene);
                the_comp_nmer->gene   = tmpgene;
                free_nmer_type( the_nmer);
475             stat->nmer[i] = NULL;
              }
              else
                die("nmer_id %d found twice\n", the_nmer-
    >nmer_id);
480         }
          }
        }
        compact_stat( stat);
      }
485

      stat_type *init_stat_type( int *membyte, conf_type
      *conf,
```

Fig. 17 (contd.)

```
                                    alph_type *alph){
490     stat_type *stat;
        int       i;


        stat = (stat_type *) salloc( 1, sizeof( stat_type));
495     *membyte         += sizeof( stat_type);
        stat->compact    = 0;

        stat->oligolen   = conf->oligolen;
        stat->alph       = alph;
500     stat->max_n_nmer = n_nmer( stat->oligolen, "acgt");
        stat->n_nmer     = 0;
        stat->nmer       =
                      (nmer_type**) salloc( stat->max_n_nmer,
    sizeof( nmer_type*));
505     membyte         += sizeof( nmer_type*) * stat-
    >max_n_nmer;
        for( i=0;i<stat->max_n_nmer;i++) {
          stat->nmer[i] = NULL;
        }
510
        stat->max_n_gene = 0;
        stat->n_gene     = 0;
        stat->gene       = NULL;

515     return stat;
    }


    nmer_type *new_nmer_type(){
520     nmer_type *the_nmer;

        the_nmer               = (nmer_type*) salloc( 1,
    sizeof( nmer_type));
        the_nmer->nmer_id      = -1;
525     the_nmer->selected     = 0;
        the_nmer->ok           = 0;
        the_nmer->lna          = NULL;
        the_nmer->n_total_gene = 0;
        the_nmer->n_gene       = 0;
530     the_nmer->max_n_gene   = 0;
        the_nmer->gene         = NULL;

        return the_nmer;
    }
535
```

Fig. 17 (contd.)

```
stat_type *read_stat( FILE *statfile, conf_type *conf,
alph_type *alph){
  char       s[256];
  int        nmer_id    = -1;
  int        gene       = -1;
  stat_type  *stat;
  nmer_type  *the_nmer;
  int        i;
  int        membyte    = 0;
  int        max_n_gene = 128;
  int        *tmpgene   = (int *)salloc( max_n_gene,
sizeof( int));
  int        n_gene     = 0;
  int        maxgene   = 0;

  stat = init_stat_type( &membyte, conf, alph);

  while( fgets(s, 254, statfile)) {
    if( s[0] == '#') {
    }
    else if( s[0] == '>') {
      if( n_gene > 0 ) {
        the_nmer->max_n_gene    = n_gene;
        the_nmer->n_total_gene  = n_gene;
        the_nmer->gene = (int *)salloc( n_gene, sizeof(
int));
        membyte       += sizeof( int) * n_gene;
        for( i=0; i<n_gene; i++) {
           the_nmer->gene[ the_nmer->n_gene] =
tmpgene[i];
           the_nmer->n_gene++;
        }
        n_gene = 0;
      }

      print_debug_interval( "Read %d nmers.", stat-
>n_nmer);

      sscanf( s, ">%d", &nmer_id);
      if( stat->n_nmer >= stat->max_n_nmer) {
        die( "Unexpected high nmer number (stat-
>n_nmer=%d) stat->max_n_nmer=%d, is the oligo length
set correctly?\n",
             stat->n_nmer, stat->max_n_nmer);
      }
      if( nmer_id >= stat->max_n_nmer) {
        die( "nmer_id (%d) over limit %d in %s, is the
oligo length set correctly?\n",
         nmer_id, stat->max_n_nmer, s);
```

Fig. 17 (contd.)

```
        }
        if( nmer_id < 0 ) {
          die( "Negative nmer_id (%d)\n", nmer_id);
        }
        stat->nmer[ nmer_id] = new_nmer_type();
        membyte          += sizeof( nmer_type);
        the_nmer = stat->nmer[ nmer_id];
        the_nmer->nmer_id     = nmer_id;
        stat->n_nmer++;
      }
      else {
        sscanf( s, "%d", &gene);
        if( n_gene >= max_n_gene) {
           max_n_gene *= 2;
           if( ! (tmpgene = realloc( tmpgene, (max_n_gene
+ 1) * sizeof( int))) )
             die( "Failed to realloc tmpgene to %d
ints\n", max_n_gene + 1);
        }
        if( gene > maxgene)
          maxgene = gene;
        tmpgene[ n_gene] = gene;
        n_gene++;
      }
    }

        if( n_gene > 0 ) {
          the_nmer->max_n_gene  = n_gene;
          the_nmer->gene = (int *)salloc(n_gene, sizeof(
int));
          membyte       += sizeof( int) * n_gene;
          for( i=0; i<n_gene; i++) {
             the_nmer->gene[  the_nmer->n_gene] =
tmpgene[i];
             the_nmer->n_gene++;
          }
          n_gene = 0;
        }

    stat->max_n_gene = maxgene + 1;
    stat->gene       = (int *)salloc( stat->max_n_gene,
sizeof( int));
    membyte          += sizeof( int) * stat->max_n_gene;
    for( i=0; i<stat->max_n_gene ; i++) {
      stat->gene[i] = 0;
    }

    printf( "# Allocated %d Mbytes for stat\n", mem-
byte/1048576);
```

Fig. 17 (contd.)

```
635     return stat;
      }


      int numcmp( const void *x, const void *y) {
640     if ( *(int *)x == *(int *)y )  return ( 0);
        else                           return ( *(int *)x >
      *(int *)y ) ? 1 : -1;
      }

645   #ifdef NODYP
      char *empty_str( int len){
        char  *str;
        str           = calloc( len+1, sizeof( char));
        memset( str, ' ', len);
650     str[len]      = 0;
        return str;
      }


655   char *reverse( char *str){
        int   i, len;
        char  c;

        len = strlen( str);
660     for( i=0; i<len/2; i++){
          c                   = str[i];
          str[i]              = str[(len-1) - i];
          str[(len-1) - i]    = c;
        }
665     return        str;
      }


      char *make_nmer( long number, int n_mer, char *alph) {
670     long  i, j, n = number;
        int  alphlen = strlen( alph);
        char *seq;

        seq = empty_str( n_mer);
675
        if( alphlen == 0 )
          strcpy( seq, "-");
        else {
          for( j=0; j<n_mer; j++){
680         i = n % alphlen;
            n = n / alphlen;
            seq[j] = alph[i];
          }
```

Fig. 17 (contd.)

```
    }
    seq = reverse( seq);

    return seq;
  }
  #endif

  char *num2nmer( long number, int oligolen, alph_type
  *alph) {
    return make_nmer( number, oligolen, alph->alph);
  }


  void sort_gene( stat_type *stat) {
    int i, j;
    nmer_type  *the_nmer;

    for( i=0; i<stat->max_n_nmer; i++) {
      if( stat->nmer[i] != NULL ) {
        the_nmer = stat->nmer[i];
        qsort( the_nmer->gene, the_nmer->n_gene, sizeof(
  int), numcmp);
      }
    }
  }


  void print_n_gene( stat_type *stat) {
    int        i, j;
    nmer_type  *the_nmer;

    for( i=0; i<stat->n_nmer; i++) {
      printf( ">%d %d %d\n", stat->nmer[i]->nmer_id,
  stat->nmer[i]->n_gene, stat->nmer[i]->selected);
    }
  }


  void remove_gene( stat_type *stat, int nmer_id, short
  select_no) {
    int i, j, k, n;
    nmer_type  *the_nmer;
    int max_n_gene;
    int *r_genes;
    int n_r_genes;
    int rgene;
    int gene;
    int *tmpgene;
```

Fig. 17 (contd.)

```
    tmpgene        = (int *) salloc( stat->max_n_gene,
  sizeof( int));
735
    r_genes        = stat->nmer[nmer_id]->gene;
    n_r_genes      = stat->nmer[nmer_id]->n_gene;
    stat->nmer[nmer_id]->selected = select_no;
    stat->nmer[nmer_id]->ok       = -3;
740
    for( i=0; i<n_r_genes; i++) {
      stat->gene[ r_genes[i]]++;
    }

745 stat->n_gene = 0;
    for( i=0; i<stat->max_n_gene; i++) {
      if( stat->gene[ i]) {
        stat->n_gene++;
      }
750 }

    for( i=0; i<stat->n_nmer; i++) {
      the_nmer = stat->nmer[i];
      if( the_nmer->ok >= 0) {
755     j=0;
        k=0;
        n=0;
        while( j<n_r_genes && k<the_nmer->n_gene ) {
          rgene = r_genes[j];
760       gene  = the_nmer->gene[k];
          if( rgene > gene ) {
            k++;
            tmpgene[ n] = gene;
            n++;
765       }
          else if( rgene < gene ) {
            j++;
          }
          else {
770         j++;
            k++;
          }
        }

775     for( ; k<the_nmer->n_gene; k++) {
          gene  = the_nmer->gene[k];
          tmpgene[ n] = gene;
          n++;
        }
780
        the_nmer->n_gene = n;
```

Fig. 17 (contd.)

```
            for( k=0; k<n; k++) {
              the_nmer->gene[k] = tmpgene[ k];
            }
785       }
        }
        free( tmpgene);
      }

790
      char nmer_ok( nmer_type *the_nmer, conf_type *conf, int
      oligolen) {

        if( the_nmer->n_total_gene   > conf->max_gene_hit) {
795       return -2;
        }

        int  self, target;
        long nmr;
800     long j;
        char *seq, *rev_seq;
        param_type     *param;
        lna_type       *best_lna;
        int  max_delta      = -99999;
805     long max_lna_id     = -1;
        int  max_self_s     = -9999;
        int  max_target_s   = -9999;
        int  max_t_delta    = -99999;
        long max_t_lna_id   = -1;
810     int  max_t_self_s   = -9999;
        int  max_t_target_s = -9999;
        int  delta;
        char ok;
        long nmer;
815     int  n_spike_pat    = pow( 2, oligolen);


        best_lna = (lna_type *) salloc( 1, sizeof(
      lna_type));
820     best_lna->lna_id       = -1;
        best_lna->self_score   = -1;
        best_lna->target_score = -1;
        best_lna->tm           = -1;

825     param = init();
        param->min_score = 2;
        param->max_res   = 1;

        rev_seq       = empty_str( oligolen);
830
```

Fig. 17 (contd.)

```
    if( conf->no_5end_spike ) {
      n_spike_pat /= 2;
    }

    for( j=0; j<n_spike_pat; j++) {
       nmer = dna2lna_spike( the_nmer->nmer_id, j);
       seq  = make_nmer( nmer, oligolen, "acgtACGT");
       calc_olig( nmer, seq, rev_seq, &self, &target, pa-
ram, 0);
       free( seq);
       delta = target - self;
       if( delta > max_delta){
         max_delta      = delta;
         max_lna_id     = nmer;
         max_self_s     = self;
         max_target_s   = target;
       }
       if( delta > max_t_delta && target > conf-
>target_min_temp){
         max_t_delta    = delta;
         max_t_lna_id   = nmer;
         max_t_self_s   = self;
         max_t_target_s = target;
       }
    }
    free_param_type( param);
    if( max_t_delta > 0) {
      best_lna->lna_id       = max_t_lna_id;
      best_lna->self_score   = max_t_self_s;
      best_lna->target_score = max_t_target_s;
      if( max_t_delta >= conf->target_minus_self )
        ok                   = 1;
      else
        ok                   = -1;
    }
    else {
      best_lna->lna_id       = max_lna_id;
      best_lna->self_score   = max_self_s;
      best_lna->target_score = max_target_s;
      ok                     = -1;
    }

    the_nmer->lna = best_lna;
    the_nmer->ok  = ok;

    return ok;
  }
```

Fig. 17 (contd.)

```
880

      int max_stat( stat_type *stat, conf_type *conf) {
        int  i;
885     int  max_n_gene      = 0;
        int  max_n_gene_prod = 0;
        int  max_nmer_id     = -1;

        if( conf->max_product) {
890       for( i=0; i<stat->n_nmer; i++) {
            if( stat->nmer[i]->ok             >= 0 &&
              stat->nmer[i]->n_gene > max_n_gene ) {
                if( stat->nmer[i]->ok == 0) {
                  nmer_ok( stat->nmer[i], conf, stat-
895   >oligolen);
                }
                if( stat->nmer[i]->ok > 0) {
                  max_n_gene   = stat->nmer[i]->n_gene;
                  max_nmer_id  = i;
900             }
            }
          }
          max_n_gene = (int) (0.80 * max_n_gene);
          for( i=0; i<stat->n_nmer; i++) {
905         if( stat->nmer[i]->ok             >= 0 &&
                stat->nmer[i]->n_gene         >  max_n_gene
      &&
            stat->nmer[i]->n_gene * stat->nmer[i]-
      >n_total_gene  >
910
      max_n_gene_prod ) {
            if( stat->nmer[i]->ok == 0) {
              nmer_ok( stat->nmer[i], conf, stat->oligolen);
            }
915         if( stat->nmer[i]->ok > 0) {
              max_n_gene_prod   =
                      stat->nmer[i]->n_gene * stat->nmer[i]-
      >n_total_gene;
              max_nmer_id  = i;
920         }
            }
          }
        }
        else {
925       for( i=0; i<stat->n_nmer; i++) {
            if( stat->nmer[i]->ok             >= 0 &&
              stat->nmer[i]->n_gene         >  max_n_gene ) {
            if( stat->nmer[i]->ok == 0) {
```

Fig. 17 (contd.)

```
            nmer_ok( stat->nmer[i], conf, stat->oligolen);
930         }
            if( stat->nmer[i]->ok > 0) {
              max_n_gene   = stat->nmer[i]->n_gene;
              max_nmer_id  = i;
            }
935         }
          }
        }
        /*
        if( max_nmer_id == -1) {
940       printf(" Found no nmers with \n");
          printf(" n_total_gene <= %d\n", conf-
      >max_gene_hit);
          printf(" n_gene > %d\n", max_n_gene);
          printf(" selected == 0\n");
        }
945     */
        return max_nmer_id;
      }

950
      int get_cover1( stat_type *stat, conf_type *conf){
        int  i, j;
        int  max_id;
        char *seq, *lnaseq;
955     int  select_no  = 0;
        char alph[10]   = "acgt";

        printf("#dnaID nmer     cover   sum tm self lnaID
      oligo\n");
960     max_id = max_stat( stat, conf);
      if(0)hi("stat->nmer[ max_id]->n_gene=%d\n", stat->nmer[
      max_id]->n_gene);
        for( i=0; i < conf->max_select &&
                  max_id >= 0 &&
965               stat->nmer[ max_id]->n_gene > 0; i++) {
          print_debug_interval( "Found %d n_mers.", i);
          if(0)print_n_gene( stat);
          select_no++;
          remove_gene( stat, max_id, select_no);
970       seq    = make_nmer( stat->nmer[ max_id]->nmer_id,
      conf->oligolen, alph);
          lnaseq = make_nmer( stat->nmer[ max_id]->lna-
      >lna_id, conf->oligolen,

975   "acgtACGT");
          printf("%6d %s %4d %5d %2d %2d %9d %s\n",
            stat->nmer[ max_id]->nmer_id,
```

Fig. 17 (contd.)

```
            seq,
            stat->nmer[ max_id]->n_gene,
980         stat->n_gene,
            stat->nmer[ max_id]->lna->target_score,
            stat->nmer[ max_id]->lna->self_score,
            stat->nmer[ max_id]->lna->lna_id,
            lnaseq);
985       free( seq);
          free( lnaseq);
          if(0) {
            for(i=0;i<57;i++){ printf("%d", stat->gene[i]); }
      printf("\n");
990         for(i=0;i<stat->nmer[max_id]->n_gene;i++){
              printf("%d ", stat->nmer[max_id]->gene[i]);
            }
            printf("\n");
          }
995
          max_id = max_stat( stat, conf);
        }

      if(0) {
1000    for(i=0;i<stat->n_nmer;i++){
          if( stat->nmer[i]->n_gene > 1 || stat->nmer[i]-
      >selected != 0) {
            printf("+ %d %d %d %d :", i,  stat->nmer[i]-
      >nmer_id, stat->nmer[i]->n_gene, stat->nmer[i]-
1005  >selected);
            for(j=0;j<stat->nmer[i]->n_gene;j++){
              printf("%d ", stat->nmer[i]->gene[j]);
            }
            printf("\n");
1010      }
        }
      }

        return select_no;
1015  }


      int nmer2num( char *seq, int len, alph_type *alph){
        int  num = 0;
1020    int  i;
        char nuc;
        int  n;
        int  mul = 1;

1025    for( i=0; i<len; i++){
          nuc = seq[len - i - 1];
```

Fig. 17 (contd.)

```
        n = strchr( alph->alph, nuc) - alph->alph;
        if( n < 0) {
          return num;
        }
        num += n * mul;
        mul *= alph->len;
      }

    return num;
  }


  void add_gene( nmer_type *the_nmer, int gene_num){
    int  *genptr = NULL;

    if( the_nmer->n_gene > 0 ) {
      genptr = bsearch( &gene_num, the_nmer->gene,
                        the_nmer->n_gene, sizeof( the_nmer-
>gene[0]),
                        numcmp);
    }

    if( genptr == NULL ) {
      the_nmer->n_gene++;
      if( the_nmer->n_gene > the_nmer->max_n_gene) {
         if( the_nmer->max_n_gene == 0) {
           the_nmer->max_n_gene = 2;
           the_nmer->gene = (int *)salloc( the_nmer-
>max_n_gene + 1,
                                           sizeof(
the_nmer->gene[0]));
         }
         else {
           the_nmer->max_n_gene *= 2;
           if( ! (the_nmer->gene = realloc( the_nmer-
>gene,
                        (the_nmer->max_n_gene + 1)* sizeof(
the_nmer->gene[0]))) )
             die( "Failed to realloc gene to %d ints\n",
the_nmer->max_n_gene);
         }
      }
      the_nmer->gene[ the_nmer->n_gene-1] = gene_num;
      /* Not necessary if gene_num is ascending
      qsort( the_nmer->gene, the_nmer->n_gene,
                     sizeof( the_nmer->gene[0]), numcmp);
      */
    }
  }
```

Fig. 17 (contd.)

```
      void add_seq2stat( char *seq, int seqlen,
                        stat_type *stat, alph_type *alph, int
1080  gene_num) {
        int  i;
        char *s, *subseq;
        int  num;
        int  end = seqlen - stat->oligolen;
1085    char stmp[255];

        for( i=0; i<=end; i++) {
          subseq = seq + i;
          num = nmer2num( subseq, stat->oligolen, alph);
1090      strncpy( stmp, subseq, stat->oligolen);
          stmp[stat->oligolen] = 0;
          /*printf( "%s %d\n", stmp, num);*/

          if( num >=  stat->max_n_nmer) {
1095        die( "%s (%d) higher than max %d\n", num, stmp,
      stat->max_n_nmer);
          }
          add_gene( stat->nmer[ num], gene_num);
          /*  $stat->[ $num]{ $seq->{ name}}++; */
1100    }
      }


      stat_type *read_fasta4stat( FILE *seqfile, conf_type
1105  *conf, alph_type *dna){
        int                 c;
        int                 i;
        char                name[255];
        char                comment[255];
1110    char                s[255];
        int                 seq_flag    = 0;
        dynamic_str_type    *seq        = NULL;
        stat_type           *stat;
        int                 membyte     = 0;
1115    int                 gene_num    = -1;
        int           n_in;

        stat = init_stat_type( &membyte, conf, dna);
        for( i=0; i<stat->max_n_nmer ;i++) {
1120      stat->nmer[i] = new_nmer_type();
          membyte         += sizeof( nmer_type);
          stat->nmer[i]->nmer_id = i;
        }
        stat->n_nmer = stat->max_n_nmer;
```

Fig. 17 (contd.)

```
1125
        while( (c = fgetc( seqfile)) != EOF){
          if( c == '>') {
            if( seq != NULL && seq->len > 0 ) {
              /*printf(">%s %s %d\n", name, comment,
1130    gene_num);*/
              add_seq2stat( seq->s, seq->len, stat, dna,
        gene_num);
              free( seq);
            }
1135        seq = new_dynamic_str_type();
            fgets( s, 254, seqfile);
            n_in = sscanf( s, "%s %s", name, comment);
            if( n_in < 2)
              comment[0]=0;
1140        if( n_in < 1)
              name[0]=0;
            gene_num++;
            seq_flag = 1;
            print_debug_interval( "Read %d genes.",
1145    gene_num);
          }
          else if( seq_flag == 1 ) {
            c = tolower( c);
            if( strchr( dna->alph, c) != NULL ) {
1150          seq = addchar( seq, c);
            }
          }
        }
        if( seq != NULL && seq->len > 0 )
1155      add_seq2stat( seq->s, seq->len, stat, dna,
        gene_num);
        free( seq);

        for( i=0; i<stat->max_n_nmer ;i++) {
1160      membyte          += sizeof( int) * stat->nmer[i]-
        >max_n_gene;
        }

        stat->max_n_gene = gene_num + 1;
1165    stat->gene       = (int *)salloc( stat->max_n_gene,
        sizeof( int));
        membyte          += sizeof( int) * stat->max_n_gene;
        for( i=0; i<stat->max_n_gene ; i++) {
          stat->gene[i] = 0;
1170    }

        printf( "# Allocated %d Mbytes for stat\n", memby-
        te/1048576);
```

Fig. 17 (contd.)

```
  return stat;
}


void print_stat( stat_type *stat) {
  int        i, j;
  nmer_type  *the_nmer;
  char       *s;
  int        str_flag = 0;

  if( verbose > 1 ) {
    str_flag = 1;

  }

  if( stat->compact ) {
    for( i=0; i<stat->n_nmer; i++) {
      print_debug_interval( "Wrote %d n_mers.", i);
        if( str_flag) {
          s = num2nmer( stat->nmer[i]->nmer_id, stat->oligolen, stat->alph);
          printf( ">%d %s\n", stat->nmer[i]->nmer_id, s);
          free( s);
        }
        else {
          printf( ">%d\n", stat->nmer[i]->nmer_id);
        }
      the_nmer = stat->nmer[i];
      for( j=0; j<the_nmer->n_gene; j++) {
        printf( "%d\n", the_nmer->gene[j]);
      }
    }
  }
  else {
    for( i=0; i<stat->max_n_nmer; i++) {
      print_debug_interval( "Wrote %d n_mers.", i);
      if( stat->nmer[i] != NULL ) {
        if( str_flag) {
          s = num2nmer( stat->nmer[i]->nmer_id, stat->oligolen, stat->alph);
          printf( ">%d %s\n", stat->nmer[i]->nmer_id, s,i);
          free( s);
        }
        else {
          printf( ">%d\n", stat->nmer[i]->nmer_id);
        }
        the_nmer = stat->nmer[i];
```

Fig. 17 (contd.)

```
            for( j=0; j<the_nmer->n_gene; j++) {
              printf( "%d\n", the_nmer->gene[j]);
            }
          }
        }
      }
    }

alph_type *new_alph( char *name, char *alph){
  alph_type   *alpht;

  alpht = (alph_type*) salloc( 1, sizeof( alph_type));
  alpht->len  = strlen( alph);
  alpht->name = ( char *) salloc( strlen( name) + 1,
sizeof( char));
  alpht->alph = ( char *) salloc( strlen( alph) + 1,
sizeof( char));
  strcpy( alpht->name, name);
  strcpy( alpht->alph, alph);

  return alpht;
}


free_alph_type( alph_type *alpht) {
  if( alpht != NULL) {
    free( alpht->name);
    free( alpht->alph);
  }
  free( alpht);
}

conf_type *new_conf_type( ){
  conf_type     *conf;
  conf = (conf_type *) salloc( 1, sizeof( conf_type));

  conf->oligolen          = 9;
  conf->fasta_in          = 0;
  conf->stat_in           = 0;
  conf->max_gene_hit_frac = 0.1;
  conf->max_gene_hit      = 999999999;
  conf->target_minus_self = 10;
  conf->target_min_temp   = 60;
  conf->complement_flag   = 0;
  conf->max_product       = 0;
  conf->max_select        = 10000;
  conf->no_5end_spike     = 0;
```

Fig. 17 (contd.)

```
  return conf;
}

int main (int argc, char* argv[]) {
  int            i;
  stat_type     *stat;
  int        n;
  alph_type     *dna = NULL;
  char           *options;
  int            next_option;
  conf_type     *conf;
  int       print_flag = 0;

  verbose = 1;

  conf = new_conf_type();

  const char* const   short_options  =
"vhl:fscrd:t:i:pnm";
  const struct option long_options[] = {
    { "verbose",              0, NULL, 'v' },
    { "help",                 0, NULL, 'h' },
    { "oligo_len",            1, NULL, 'l' },
    { "fasta_in",             0, NULL, 'f' },
    { "stat_in",              0, NULL, 's' },
    { "complement",           0, NULL, 'c' },
    { "max_product",          0, NULL, 'm' },
    { "no_5end_spike",        0, NULL, 'n' },
    { "target_minus_self",    1, NULL, 'd' },
    { "target_min_temp",      1, NULL, 't' },
    { "max_gene_hit_frac",    1, NULL, 'i' },
    { "dump_stat",            0, NULL, 'p' },
    { NULL,                   0, NULL,  0  }   /* Re-
quired at end of array.  */
  };

  program_name = argv[0];
  run_time( TIME_INIT);

  do {
    next_option = getopt_long (argc, argv,
short_options,
        long_options, NULL);
    switch (next_option)
    {
    case 'l':   /* -l or --oligo_len*/
```

Fig. 17 (contd.)

```
            options = optarg;
            conf->oligolen = atoi( options);
            break;
          case 't':   /* -t or --target_min_temp*/
            options = optarg;
            conf->target_min_temp = atoi( options);
            break;
          case 'd':   /* -d or --target_minus_self*/
            options = optarg;
            conf->target_minus_self = atoi( options);
            break;
          case 'i':   /* -i or --max_gene_hit_frac*/
            options = optarg;
            conf->max_gene_hit_frac = atof( options);
            break;
          case 'f':   /* -f or --fasta_in*/
            conf->fasta_in = 1;
            break;
          case 's':   /* -s or --stat_in*/
            conf->stat_in = 1;
            break;
          case 'v':   /* -v or --verbose*/
            verbose++;
            break;
          case 'p':   /* -p or --dump_stat*/
            print_flag = 1;
            break;
          case 'c':   /* -c or --complement*/
            conf->complement_flag = 1;
            break;
          case 'm':   /* -m or --max_product*/
            conf->max_product = 1;
            break;
          case 'n':   /* -n or --no_5end_spike*/
            conf->no_5end_spike = 1;
            break;
          case 'h':   /* -h or --help*/
            usage("");
            break;
          }
        }
        while (next_option != -1);


        dna = new_alph( "dna", "acgt");

        if( verbose >= 1 ) {
          printf( "# Parameters:\n");
```

Fig. 17 (contd.)

```
1370      printf( "# oligo length      = %d\n", conf-
      >oligolen);
          printf( "# max_gene_hit_frac = %f\n", conf-
      >max_gene_hit_frac);
          printf( "# target_minus_self = %d\n", conf-
1375  >target_minus_self);
          printf( "# target_min_temp   = %d\n", conf-
      >target_min_temp);
          printf( "# complement_flag   = %d\n", conf-
      >complement_flag);
1380      printf( "# max_product       = %d\n", conf-
      >max_product);
          printf( "# no_5end_spike_flag = %d\n", conf-
      >no_5end_spike);
          printf( "# max_select        = %d\n", conf-
1385  >max_select);
        }

        if( conf->fasta_in) {
          printf("# Scanning fasta file\n");
1390      stat = read_fasta4stat( stdin, conf, dna);
          run_time( LAP_TIME);
          printf("# Read %d nmer statistics in %d s\n", stat-
      >n_nmer, t_run);
        }
1395    else if( conf->stat_in) {
          printf("# Reading statistics\n");
          stat = read_stat( stdin, conf, dna);
          run_time( LAP_TIME);
          printf("# Read %d nmer statistics in %d s\n", stat-
1400  >n_nmer, t_run);
        }
        else
          usage( "Please use either fasta or stat as input
      data.");
1405    printf( "# Found %d genes\n", stat->max_n_gene);
        conf->max_gene_hit = (conf->max_gene_hit_frac * stat-
      >max_n_gene);
        printf( "# max_gene_hit      = %d\n", conf-
      >max_gene_hit);
1410
        printf("# Sorting genes\n");
        sort_gene( stat);
        run_time( LAP_TIME);
        printf("# Genes sorted in %d s\n", t_run);
1415
        if( print_flag ) {
          compact_stat( stat);
          print_stat( stat);
```

Fig. 17 (contd.)

```
    }
1420  else {

      if( conf->complement_flag){
        printf("# Join complement nmers\n");
        comp_stat( stat);
1425    run_time( LAP_TIME);
        printf("# Joined nmers in %d s\n", t_run);
      }
      else {
        compact_stat( stat);
1430  }

      printf("# Get cover\n");
      n = get_cover1( stat, conf);
      run_time( LAP_TIME);
1435  printf("# Got %d nmer cover in %d s\n", n, t_run);

      if(0)print_n_gene( stat);
    /* print_stat( stat); */
    }
1440
    /*free_stat( stat); */
    free( conf);

    run_time( TOTAL_TIME);
1445 printf("# Run time %d s\n", t_run);
    exit(0);
  }


1450
```

Program dyp.c

```
/*

 #$Id: dyp.c,v 1.5 2003/04/23 09:33:50 nt Exp $
1455
DESCRIPTION
      Loading     sequences:
      -f fastafile
      -seq sequence
1460
      Set min_score to 0 to see all structures
      -min_score min_score

      Size of     sliding     window default is INT_MAX
1465  -depth depth

      Number of results to show
```

Fig. 17 (contd.)

```
            -max_res max_res

1470          -secondary_structure
            Calculates the secondary structure of an oligo

            -self_annealing
            Calculates the binding energy between an oligo
1475  and
            it self.

            -hybridization
            Calculates the binding energy between two differ-
1480  ent
            oligos,     or an oligo and   its target.

      AUTHOR
            Copyright Niels   Tolstrup 2002 and 2003
1485        Exiqon
      */


      #include <stdio.h>     /* fprintf, rand..*/
1490  #include <stdlib.h>    /* calloc..   */
      #include <stdarg.h>    /* va_list..  */
      #include <string.h>    /* strlen..   */
      #include <limits.h>    /* USHRT_MAX..      */
      #include <math.h>      /* pow        */
1495  #include <ctype.h>     /* tolower         */
      #include "getopt.h"    /* getopt_long      */
      #ifdef LIB
      #include "dyp.h"       /* param_t..       */

1500  #endif

      /* export MALLOC_TRACE=dyp_memtrace
         mtrace dyp $MALLOC_TRACE
         #include <mcheck.h>    mtrace
1505  */

      /* memory debugging with dmalloc
       * Change Makefile:
       * LIBS = -ldmalloc
1510   * CFLAGS = -g -Wall -lm -
      L/home/sites/site2/users/lnatools/web/hybridization/src
      /dmalloc/dmalloc-4.8.2 -DDMALLOC -
      I/home/sites/site2/users/lnatools/web/hybridization/src
      /dmalloc/dmalloc-4.8.2
1515   *
       * run:
```

Fig. 17 (contd.)

```
 * ../dmalloc/dmalloc-4.8.2/dmalloc -b -l
/home/sites/site2/users/lnatools/web/hybridization/src/
dyp-0.0.1/dmalloc.log -i 10 low -p check-fence -p
check-heap
 *
 * compile and run and check dmalloc.log
 *
   #include "dmalloc.h"
 */




/********************** DEFINES ***************/

#define THEVERSION "1.3 2002-11-06-13-52"
#define     TRACEBACK_INT unsigned short
#define     TRACEBACK_INT_MAX USHRT_MAX

#define HI printf( "Hello\n");

#define     SECONDARY_STRUCTURE 1
#define     SELF_ANEALING 2
#define     HYBRIDIZATION 4
#define     TM 8

#define ALLOLIG 1
#define CLUSTER 2

/* Increase the   stack by this amount in case of
      overflow */
#define     STACK_CHUNK   255

/* Number of sequences to allocate space for before re-
alloc */
#define     SEQS_CHUNK    255

/* Size     of sequence to allocate space for before re-
alloc */
#define     SEQ_CHUNK     1024

/* Longest possible line in a matrix  that can be read
*/
#define     MAX_LINE_LEN  5000

#define MIN(a,b) (a<b?a:b)
#define MAX(a,b) (a>b?a:b)
```

Fig. 17 (contd.)

```
/*********** GLOBAL VARIABLES ***************/


int         verbose     = 0;
char*       program_name;



/************ STRUCTURES ********************/


#ifndef LIB
typedef     struct sequences_t {
       int        nseq;
       int        maxnseq;
       char       **seqs;
      } sequences_type;

typedef     struct dynamic_str_t {
       int        len;
       int        maxlen;
       char       *s;
      } dynamic_str_type;
#endif



typedef     struct pair_t {
       TRACEBACK_INT          i;
       TRACEBACK_INT          j;
      } pair_type;

typedef     struct stack_t {
       TRACEBACK_INT          sp;
       TRACEBACK_INT          size;
       pair_type        *stack;
      } stack_type;


typedef     struct tokenarray_t {
        int ntok;
        int maxntok;
        char      **tok;
      } tokenarray_type;
```

Fig. 17 (contd.)

```
typedef     struct matrix_t   {
       char *seq1;
       char *seq2;
       int  l1;
       int  l2;
       int  max_size;
       int  *d;

      } matrix_type;


#ifndef LIB
typedef     struct score_rec_t {
       int        alph_len;
       int        gap_start;
       int        gap_cont;
       int        mismatch_cont;
       int        loop_score;
       int        match_cont_factor;
       int        match_threshold;
       int        min_strong_ident_score;
       int        min_ident_score;
       int        min_sim_score;
       char       *alph;
       int        *mat;
      } score_rec_type;


typedef     struct param_t {
       char       *alph;
       int        *mathyp;
       int        depth;
       int        global;
       int        min_score;
       int        max_res;
       sequences_type   *seqs;
       sequences_type   *seqs2;
       score_rec_type   *score_rec;
     } param_type;
#endif


/**************   SYSTEM ****************/


void die( char *fmt, ...){
  va_list ap;
  fprintf( stderr, "Uhoh: ");
  va_start( ap,   fmt);
```

Fig. 17 (contd.)

```
  vfprintf( stderr, fmt, ap);
  va_end( ap);
  fprintf( stderr, "\n");
  if(1)exit(     1);
}

#ifndef LIB
void usage( char *fmt, ...){
  va_list ap;
  va_start( ap,   fmt);
  vfprintf( stderr, fmt, ap);
  va_end( ap);
  fprintf( stderr,
      "\nUsage: dyp -h,--help\n"
      "       -v,--verbose\n"
      "       -r,--version\n"
      "       -u,--secondary_structure\n"
      "       -a,--self_anealing\n"
      "       -y,--hybridization\n"
      "       -t,--tm\n"
      "       -o,--output filename\n"
      "       -i,--input filename\n"
      "       -j,--input2 filename2\n"
      "       -s,--seq sequence\n"
      "       -z,--seq2 sequence2\n"
      "       -d,--depth depth\n"
      "       -g,--global\n"
      "       -n,--min_score score\n"
      "       -p,--max_res   n\n"
      "       -m,--matrix matfn\n"
      "       -l,--allolig length_of_oligo\n"
      "       -w,--n_mer_range nmer_start-
nmer_end[,spike_start-spike_end]\n"
      "       -c,--cluster score_cutoff\n"
      "       -k,--n_okseq include_first_n_seqs\n"
      "       -f,--spikefreq freq\n"
      "       -e,--sample n_samples\n"
      "\n"
      "DESCRIPTION\n"
      "     n_mer_range is used with allolig to specify
a subset"
      "     if only nmer_start and nmer_end are given
lna numbering is used"
      "     if the spike range is given, dna numbering
is used");
  fprintf( stderr, "\n");
  exit(     1);
}
```

Fig. 17 (contd.)

```
#endif

1715
void hi( char *fmt, ...){
  va_list ap;
  fprintf( stdout, "Hi there " );
  va_start( ap,   fmt);
1720  vfprintf( stdout, fmt, ap);
  va_end( ap);
  fprintf( stdout, "\n");
  fflush( stdout);
}
1725

void *salloc( int nobj, int size){
  void *mem;
  /*mem =   calloc(     nobj, size);*/
1730  mem =     malloc(     nobj * size);
  /*printf( "calloc: allocating     %d objects of size
%d bytes\n",      nobj, size);*/
  if( mem == NULL){
    die( "Could   not allocate %d   x %d bytes\n", nobj,
1735 size);
  }
  return mem;
}

1740

FILE *sfopen( char *fn, const char *mode){
  FILE *file;
  file = fopen(   fn, mode);
  if( file == NULL )
1745      die( "Could not   open %s     for %s.", fn, mode);
  return file;
}


1750 int check_limits(){
  int int_bits;
  int long_bits;

  int_bits  = log( UINT_MAX)/ log( 2) + 0.5;
1755  long_bits = log( ULONG_MAX)/ log( 2) + 0.5;
  if( int_bits != 32 ) {
    die( "Int must be 32 bits, but is %d\n", int_bits);
  }
  if( long_bits != 32 ) {
1760    die( "Long must be 32 bits, but is %d\n",
long_bits);
```

Fig. 17 (contd.)

```
1765
      void check_endian(){
        long n = 1;
        if( n >> 1) {
          /* Can this happen ???? */
1770      die( "Endian issue\n");
        }
      }


1775  /********* STRING FUNCTIONS ***********/


      char *empty_str( int len){
        char      *str;
1780    str       = calloc( len+1, sizeof( char));
        memset( str, ' ', len);
        str[len]  = 0;
        return str;
      }
1785

      char *reverse( char *str){
        int i, len;
        char      c;
1790
        len = strlen( str);
        for( i=0; i<len/2; i++){
          c             = str[i];
          str[i]        = str[(len-1) - i];
1795      str[(len-1) - i]    = c;
        }
        return    str;
      }

1800
      char *lower( char *s) {
        int i, l = strlen( s);
        for( i=0; i<l; i++)
          s[i] = tolower( s[i]);
1805    return s;
      }


      char tocomp( char c) {
1810    char fr_s[35] = "acgtumrwsykvhdbxnACGTUMRWSYKVHDBXN";
```

Fig. 17 (contd.)

```
  char to_s[35] = "tgcaakywsrmbdhvxxTGCAAKYWSRMBDHVXX";
  char *p;
  char cc;

  p  = strchr( fr_s, c);
  if( p ) {
    p  = to_s + (p - fr_s);
    cc = *p;
  }
  else
    cc = 'x';
  return cc;
}

char *comp( char *s) {
  int i, l = strlen( s);
  for( i=0; i<l; i++)
    s[i] = tocomp( s[i]);
  return s;
}

/*********** DYNAMIC STRINGS *********/

void free_dynamic_str_type(   dynamic_str_type *seq) {
  free(     seq->s);
  free(     seq);
}

dynamic_str_type *new_dynamic_str_type(){
    dynamic_str_type *seq;

    seq     = calloc( 1, sizeof( dynamic_str_type));
    seq->len        = 0;
    seq->maxlen     = SEQ_CHUNK;
    seq->s    = calloc( seq->maxlen + 1, sizeof(
char));
    seq->s[0]       = '\0';
    return seq;
}

dynamic_str_type *addchar( dynamic_str_type *seq, int
c) {
  if( seq == NULL) {
    seq     = new_dynamic_str_type();
```

Fig. 17 (contd.)

```
1860    }
        if( seq->len + 1 >= seq->maxlen ) {
          seq->maxlen    += SEQ_CHUNK;
          if(     ! (seq->s = realloc( seq->s, (seq->maxlen +
      1) * sizeof( char))) )
1865          die( "Failed to realloc seq to %d chars", seq-
      >maxlen);
        }
        seq->s[ seq->len++] = c;
        return seq;
1870  }


      /******* I/O of   strings     *******/

1875

      void add_seq2seqs( char *seq, sequences_type *seqs){
        int new_size, i;

        if( seqs->nseq < seqs->maxnseq ) {
1880      seqs->seqs[ seqs->nseq++] = seq;
        }
        else {
          new_size = (seqs->maxnseq + SEQS_CHUNK) * sizeof(
      char *);
1885      if( (seqs->seqs = realloc( seqs->seqs, new_size)))
      {
            seqs->maxnseq = seqs->maxnseq + SEQS_CHUNK;
            for( i=seqs->nseq; i<seqs->maxnseq; i++)
               seqs->seqs[i] = NULL;
1890        seqs->seqs[ seqs->nseq++] = seq;
          }
          else {
            die( "Failed to realloc seq array to %d bytes",
      new_size);
1895      }
        }
      }


1900  sequences_type *new_sequences_type( int maxnseq){
        int i;
        sequences_type *seqs;
        seqs             = calloc( 1, sizeof( se-
      quences_type));
1905    seqs->nseq       = 0;
        seqs->maxnseq    = maxnseq;
        seqs->seqs       = calloc( seqs->maxnseq, sizeof( char
      *));
```

Fig. 17 (contd.)

```
    for( i=0; i<seqs->maxnseq; i++)
1910    seqs->seqs[i] = NULL;

    return seqs;
}

1915 void free_sequences_type( sequences_type *seqs) {
    free( seqs->seqs);
    free( seqs);
}

1920
void print_lines( FILE *outfile, char **str, int nstr){
    int linelen =   80;
    int i, j, p, len;

1925    if( nstr > 0 ) {
      len     = strlen( str[0]);

      for( j=0; j<nstr; j++) {
        if( strlen( str[j]) != len ) {
1930      die( "String len = %d differs from first string
          len %d %s\n",
                strlen(     str[j]), len, str[j]);
        }
      }
1935    p =     0;

      while( p < len) {
        for( j=0;   j<nstr;     j++) {
        for( i = p; i <   p + linelen && i < len; i++) {
1940      putc( str[j][i], outfile);
        }
        putc( '\n', outfile);
        }
        p     += linelen;
1945    putc( '\n', outfile);
      }
    }
}

1950
void print_fasta( FILE *outfile, sequences_type *seqs)
{
    int i;
    char *seq;
1955
    for( i=0; i<seqs->nseq; i++) {
      fprintf( outfile, ">seq_%d\n", i);
```

Fig. 17 (contd.)

```
        seq = seqs->seqs[i];
        print_lines( outfile, &seq, 1);
    }
  }


  void read_fasta( FILE *seqfile,     sequences_type
  *seqs, char *alph){
    int           c;
    char                name[255];
    char                comment[255];
    char                s[255];
    int           seq_flag      = 0;
    dynamic_str_type  *seq      = NULL;

    while( (c = fgetc( seqfile)) != EOF){
      if(     c == '>') {
        if( seq != NULL && seq->len > 0 )   {
        add_seq2seqs( seq->s, seqs);
        free( seq);
        seq = new_dynamic_str_type();
        }
        fgets( s,   254, seqfile);
        sscanf( s, "%s %s", name, comment);
        seq_flag = 1;
      }
      else if( seq_flag == 1 ) {
        if( strchr( alph, c) != NULL ) {
        seq = addchar( seq, c);
        }
      }
    }
    if( seq != NULL && seq->len >     0 )
      add_seq2seqs( seq->s, seqs);

    free(     seq);
  }


  /********* STACK *************/


  stack_type *init_stack( TRACEBACK_INT size) {
    stack_type *stack;

    stack           = calloc( 1, sizeof( stack_type));
    stack->sp = 0;
    stack->size     = size;
```

Fig. 17 (contd.)

```
  stack->stack    = calloc( size,   sizeof(
      pair_type));

  return stack;
}


void free_stack( stack_type *stack){
  free(     stack->stack);
  free(     stack);
}


void print_pair( pair_type pair){
  printf( "%4d,   %4d\n",     pair.i,     pair.j);
}


void push( stack_type *stack, pair_type  pair) {
    int     new_size;
    if(     stack->sp < stack->size ) {
      stack->stack[ stack->sp++] = pair;
      if(0){printf("pushed :"); print_pair( pair);}
    }
    else {
      new_size = (stack->size + STACK_CHUNK) * sizeof(
pair_type);
      if( (stack->stack = realloc( stack->stack,
new_size))) {
          stack->size                = stack->size +
STACK_CHUNK;
          stack->stack[ stack->sp++] = pair;
      }
      else {
          die( "Failed to realloc stack to %d  bytes",
      new_size);
      }
    }
}


pair_type setpair( int i, int j) {
  pair_type pair;
  pair.i = i;
  pair.j = j;
  return pair;
}
```

Fig. 17 (contd.)

```
pair_type pop( stack_type *stack) {
  if( stack->sp > 0 )
    return stack->stack[      --(stack->sp)];
  else
    return setpair( TRACEBACK_INT_MAX, TRACE-
BACK_INT_MAX);
}


void test_stack(){
  stack_type *stack;
  int i;

  stack     = init_stack( STACK_CHUNK);

  for( i=0; i<30; i++)
   push( stack,   setpair( i, i+1));

  for( i=0; i<30; i++)
   print_pair( pop( stack));

  free_stack( stack);
}


/************* OLIGO FUNCTIONS ********/


long n_nmer( int n_mer, char *alph){
  int alphlen = strlen( alph);
  long n_nmer  = 0;

  if( alphlen == 0 )
    n_nmer = 0;
  else
    n_nmer = pow( alphlen, n_mer);

  return n_nmer;
}


long big_nmer( int n){
  int i;
  long n_tmp;
  long mask = 1;
  long res = 0;

  for( i=0;i<16;i++){
    n_tmp  = n & mask;
```

Fig. 17 (contd.)

```
    n_tmp  = n_tmp << 3;
    res   += n_tmp;
    mask   = mask << 1;
  }

  return res;
}


char *make_nmer( long number, int n_mer, char *alph) {
  long  i, j, n = number;
  int  alphlen = strlen( alph);
  char *seq;

  seq = empty_str( n_mer);

  if( alphlen == 0 )
    strcpy( seq, "-");
  else {
    for( j=0; j<n_mer; j++){
      i = n % alphlen;
      n = n / alphlen;
      seq[j] = alph[i];
    }
  }
  seq = reverse( seq);

  return seq;
}


int rando( int max) {
  int r;
  do
    r = (rand() * max)/RAND_MAX;
  while( r==max);
  return r;
}


char *make_random_nmer( int n_mer, int spikefreq, char
*alph) {
  long  i, j;
  int  alphlen = strlen( alph);
  int  halflen = alphlen / 2;
  char *seq;
  char phase   = 0;

  seq = empty_str( n_mer);
```

Fig. 17 (contd.)

```
    if( spikefreq)
      phase = rando( spikefreq);


    if( alphlen == 0 )
      strcpy( seq, "-");
    else {
      if( spikefreq)
        for( j=0; j<n_mer; j++){
        i = rando( halflen);
          if( ( (j+phase) % spikefreq) == 0)
          i += halflen;
        seq[j] = alph[i];
        }
      else
        for( j=0; j<n_mer; j++){
        i = rando( alphlen);
        seq[j] = alph[i];
        }
    }
    return seq;
}



/************* MATRIX I/O ************/

score_rec_type *init_score_rec_type( int gap_start, int
      gap_cont, int mismatch_cont,
                          int loop_score,   int
match_threshold,
                          int match_cont_factor,
                          int min_strong_ident_score,
                          int min_ident_score, int
min_sim_score,
                          char *alph, int   *mat){
  score_rec_type *score_rec;

  score_rec        = (score_rec_type *) calloc( 1,
sizeof( score_rec_type));
  score_rec->alph_len         = strlen( alph);
  score_rec->gap_start        = gap_start;
  score_rec->gap_cont         = gap_cont;
  score_rec->mismatch_cont    = mismatch_cont; /* a
negativ value turn this feature off */
  score_rec->loop_score       = loop_score;
  score_rec->match_threshold  = match_threshold;
```

Fig. 17 (contd.)

```
  score_rec->match_cont_factor      = match_cont_factor;
  score_rec->alph      = alph;
  score_rec->mat       = mat;
  score_rec->min_strong_ident_score =
min_strong_ident_score;
  score_rec->min_ident_score  = min_ident_score;
  score_rec->min_sim_score    = min_sim_score;

  return score_rec;
}


void free_score_rec_type( score_rec_type *score_rec){
  free(     score_rec->alph);
  free(     score_rec->mat);
  free(     score_rec);
}


void free_param_type( param_type *param){
  free_score_rec_type( param->score_rec);
  free_sequences_type( param->seqs);
  free_sequences_type( param->seqs2);
  free(     param);
}


void validate_score_rec_type( score_rec_type
*score_rec, int min, int max){
  int i, j;
  if( strlen( score_rec->alph) < 1) { die( "No alpha-
bet\n"); }
  if( strlen( score_rec->alph) != score_rec->alph_len)
      {
    die( "alph length mismatch %d,%d\n", strlen(
score_rec->alph),
                                          score_rec-
>alph_len);
  }
  for( i=0;i<score_rec->alph_len;i++){
    for( j=0;j<score_rec->alph_len;j++){
      if( score_rec->mat[i*score_rec->alph_len+j]<min )
      die( "Matrix out of range min=%d", min);
      if( score_rec->mat[i*score_rec->alph_len+j]>max )
      die( "Matrix out of range %d max=%d",
                  score_rec->mat[i*score_rec-
>alph_len+j], max);
    }
  }
```

Fig. 17 (contd.)

```
    if( score_rec->gap_start  > max ) die( "gap_start out
        of range max");
    if( score_rec->gap_start  < min ) die( "gap_start out
        of range max");
    if( score_rec->gap_cont   > max ) die( "gap_cont out
of range    max");
    if( score_rec->gap_cont   < min ) die( "gap_cont out
of range    max");
    if( score_rec->loop_score > max ) die( "loop_score
out of range max");
    if( score_rec->loop_score < min ) die( "loop_score
out of range max");
    }


void print_mat(   char *seq1, char *seq2, int *mat){
    int i, j;
    int len1  =     strlen(     seq1);
    int len2  =     strlen(     seq2);
    int p_num =     1;

    if( p_num == 1) {
      printf( "       ");
      for( i=0;i<len1;i++){
        printf( " %2d   ", i);
      }
      printf( "\n   ");
    }
    printf( "  ");
    for( i=0;i<len1;i++){
      printf( "    %c ", seq1[i]);
    }
    printf( "\n\n");
    for( j=0;j<len2;j++){
      if(     p_num == 1) {
        printf( "%2d %c ", j, seq2[j]);
      }
      else {
        printf( "%c ", seq2[j]);
      }
      for( i=0;i<len1;i++){
        printf( "%5d ", mat[j*len1+i]);
      }
      printf( "\n");
    }
}

/******* TOKENIZE A LINE ************/
```

Fig. 17 (contd.)

```
2300

      void addtoken( char* token, tokenarray_type *tokena){
        tokena->ntok++;
        if( tokena->ntok > tokena->maxntok ) {
2305      tokena->maxntok += 100;
          if(     ! (tokena->tok =
                  realloc( tokena->tok, tokena->maxntok *
            sizeof(     char)))     )
              die( "Failed to realloc tokenarray to  %d to-
2310  kens", tokena->maxntok);
        }
        tokena->tok[ tokena->ntok - 1] = token;
      }

2315

      tokenarray_type   *alloc_tokenarray_type(){
        tokenarray_type *tokena;

        tokena      = calloc( 1, sizeof( tokenarray_type));
2320    tokena->maxntok = 30;
        tokena->ntok      = 0;
        tokena->tok       = calloc( tokena->maxntok, sizeof(
      char*));
        return tokena;
2325  }


      void free_tokenarray_type( tokenarray_type *tokena){
        free(     tokena->tok);
2330    free(     tokena);
      }


      void tokenize( char *s, tokenarray_type *tokena){
2335    char            *token;

        tokena->ntok    = 0;
        token     = strtok( s, " \n");
        if( token )
2340      addtoken( token, tokena);
        while( (token = strtok( NULL, " \n")))
          addtoken( token, tokena);
      }

2345

      int tokenarray_is_num_list( tokenarray_type *tokena){
        int  i = 0;
        char *endp;
```

Fig. 17 (contd.)

```
  int  res = 0;

  while( i<tokena->ntok && strtol( tokena->tok[i],
&endp, 10) == i) {
    if(     *endp != 0 )
      res = 0;
    i++;
  }
  if( i     == tokena->ntok)
    res     = i;
  else
    res     = 0;
  return res;
}


char *tokenarray2str( tokenarray_type *tokena){
  int i=0;
  char *alph;

  alph = calloc( tokena->ntok +     1, sizeof( char));
  for( i=0; i<tokena->ntok; i++) {
    if(     strlen(     tokena->tok[i])   != 1)
      die( "Sorry, the word %s was found were a char was
expected.",
                                              tokena-
>tok[i]);
    alph[i] = *tokena->tok[i];
  }
  alph[i+1] = 0; /* EOL */

  return alph;
}


int tokenarray2dat( tokenarray_type *tokena, int
with_num,
                    matrix_type *mt, char *c){
  int  i = 0;
  int  j = 0;
  char *endp;
  int  err = 0;
  int  *dat;

  if( with_num ) {
    if(     atoi( tokena->tok[i]) != mt->l2)
      err = 1;
    i++;
  }
```

Fig. 17 (contd.)

```
        if( strlen(tokena->tok[i]) !=     1)
          err     = 2;
2400    *c = *tokena->tok[i];
        i++;

        if( tokena->ntok - i != mt->l1 )
          err     = 3;
2405    else {
          dat     = calloc( mt->l1, sizeof( int));
          for( ; i<tokena->ntok; i++)     {
            dat[j] = strtol( tokena->tok[i], &endp, 10);
            if( *endp   != 0 )
2410        err = j     + 10;
            j++;
          }
        }
        if( !err) {
2415      if(     mt->l1 * mt->l2   > mt->max_size ) {
            mt->max_size += 100;
            if( ! (mt->d =
                  realloc( mt->d,   mt->max_size * sizeof(
      int))) )
2420          die( "Failed to realloc matrix to %d int's",
      mt->max_size);
          }
          for( j=0; j<mt->l1; j++)
            mt->d[mt->l1 * mt->l2 + j] = dat[j];
2425      mt->l2++;
        }
        free( dat);
        return err;
      }
2430

      /*********** I/O score_rec_type     ************/


2435  matrix_type *read_mat( FILE *file){
        char      *s, *res;
        int n = MAX_LINE_LEN;
        int col_name_found = 0;
        int numbers_found  = 0;
2440    tokenarray_type *tokena;
        dynamic_str_type  *seq    = NULL;
        char      c;
        matrix_type  *mt;

2445    mt        = calloc( 1, sizeof( matrix_type));
        mt->max_size    = 100;
```

Fig. 17 (contd.)

```
        mt->d           = calloc( mt->max_size, sizeof(
            int));
        mt->l2    = 0;
2450
        tokena    = alloc_tokenarray_type();

        s         = calloc( n + 1, sizeof( char));
        seq       = new_dynamic_str_type();
2455
        do {
          res     = fgets( s, n +   1, file);
          if(     res != NULL ) {
           tokenize( s, tokena);
2460       /*for(i=0;i<tokena->ntok;i++) printf( "%d %s\n",
      i, tokena->tok[i]);*/
           if( !col_name_found ) {
            /* Is it a column number line? */
            if( !numbers_found && (mt->l1 = tokenar-
2465  ray_is_num_list( tokena)))    {
           numbers_found =   1;
           }
           else {
           /* Is it a column name line? */
2470       mt->seq1  =  tokenarray2str( tokena);
           if( numbers_found ) {
             if( mt->l1   != strlen( mt->seq1))
               die("Expected a string of length %d  but
      found %s", mt->l1,
2475                                                    mt-
      >seq1);
           }
           else
             mt->l1    = strlen( mt->seq1);
2480       col_name_found = 1;
           }
          }
          else {
           /* skip empty lines */
2485       if( tokena->ntok > 0) {
           /* Is it a data line? */
            if( tokenarray2dat( tokena, numbers_found, mt,
      &c) == 0)   {
           seq = addchar(   seq, c);
2490       }
           else {
           res = NULL;
           }
          }
2495      }
```

Fig. 17 (contd.)

```
        }
      } while( res);

      mt->seq2 = calloc( strlen( seq->s) + 1, sizeof(
2500  char));

      strcpy( mt->seq2, seq->s);
      free_dynamic_str_type( seq);

2505  free(     s);
      free_tokenarray_type( tokena);
      return mt;
    }

2510
    void print_score_rec_type( score_rec_type *score_rec){
      printf( "alph_len   %d\n", score_rec->alph_len);
      printf( "gap_start  %d\n", score_rec->gap_start);
      printf( "gap_cont   %d\n", score_rec->gap_cont);
2515  printf( "alph         %s\n", score_rec->alph);
      print_mat( score_rec->alph, score_rec->alph,
    score_rec->mat);
    }

2520
    void read_score_rec_type( char *fn, score_rec_type
    *score_rec){
      FILE      *file;
      matrix_type *mt;
2525
      file = sfopen( fn, "r");
      fscanf( file,   "alph_len   %d\n", &(score_rec-
    >alph_len));
      fscanf( file,   "gap_start  %d\n", &(score_rec-
2530  >gap_start));
      fscanf( file,   "gap_cont   %d\n", &(score_rec-
    >gap_cont));
      fscanf( file,   "min_strong_ident_score   %d\n",
    &(score_rec->min_strong_ident_score));
2535  fscanf( file,   "min_ident_score   %d\n", &(score_rec-
    >min_ident_score));
      fscanf( file,   "min_sim_score   %d\n", &(score_rec-
    >min_sim_score));
      fscanf( file,   "alph     %s\n", score_rec->alph);
2540  mt = read_mat( file);
      if( mt->l1 !=  mt->l2)
        die("Matrix   not quadratic %d %d", mt->l1, mt->l2);
      if( strcmp( mt->seq1, mt->seq2) != 0)
```

Fig. 17 (contd.)

```
      die("Matrix   sequences not identical %s %s",
2545        mt->seq1, mt->seq2);
    if( strcmp( mt->seq1, score_rec->alph) != 0)
      die("Matrix   alphabet differs from alph %s %s", mt-
>seq1, score_rec->alph);
    if( score_rec->mat )
2550      free( score_rec->mat);
    score_rec->mat = mt->d;
    free( mt->seq1);
    free( mt->seq2);
    free( mt);
2555  fclose( file);
}


/******** MATRIX *******/
2560

int *seq2idx_seq( char *seq, score_rec_type *score_rec)
      {
  char      c[2];
2565  int i, l;
  int *idx_seq;
  int id;

  l      = strlen( seq);
2570  idx_seq  = calloc( l, sizeof(     int));

  for( i=0; i<l; i++) {
    c[0] = seq[i];
    c[1] = 0;
2575    id       = strcspn( score_rec->alph, c);
    if(     id >= score_rec->alph_len) {
      fprintf( stderr,
      "Sorry,     the character %c was not found in the
alphabet %s\n",
2580                                    seq[i],
      score_rec->alph);
     id     = score_rec->alph_len - 1;
    }
    idx_seq[i]      = id;
2585  }
  return idx_seq;
}


2590 int matidx( int   i, int j, int l1, int l2){
  if( i<0 ) die( "matrix index      i (%d) less than
zero",      i);
```

Fig. 17 (contd.)

```
  if( i>=l1 )     die( "matrix index i (%d)     too
large (%d)", i, l1);
  if( j<0 ) die( "matrix index      j (%d) less than
zero",      j);
  if( j>=l2 )     die( "matrix index j (%d)     too
large (%d)", j, l2);
  return    j*l1 + i;
}


int matsize( int l1, int l2){
  return    l1*l2;
}


int m( int i, int j, matrix_type *mat) {
  return    mat->d[     matidx(     i, j, mat->l1, mat-
>l2)];
}


void print_annot( char *seq, char *annot){
  printf("%s\n", seq);
  printf("%s\n", annot);
}


char match_sym(   char a,     char b,     score_rec_type
*score_rec) {
  int i1, i2;
  char      *s;
  int score;

    s = strchr( score_rec->alph, a);
    if(     s == NULL) return ' ';
    i1      = s -  score_rec->alph;
    s = strchr( score_rec->alph, b);
    if(     s == NULL) return ' ';
    i2      = s -  score_rec->alph;

    score = score_rec->mat[ i2*score_rec->alph_len+i1];

  if( score >= score_rec->min_strong_ident_score )
    return '|';
  else if( score >= score_rec->min_ident_score )
    return ':';
  else if( score >= score_rec->min_sim_score )
    return '.';
```

Fig. 17 (contd.)

```
  return ' ';
}

/************* ALIGN - self association
****************/


void align_global_traceback( FILE *outfile, param_type
*param,
                             char *seq1, char *seq2, int
*path){
  int  l1    = strlen( seq1);
  int  l2    = strlen( seq2);
  int  done  = 0;
  int  is    = 0;
  int  i, j;
  int  i1, i2;
  char *qseq, *mseq, *tseq;

  if( l1>0 && l2>0 ) {

    qseq = calloc( l1+l2+1, sizeof( char));
    mseq = calloc( l1+l2+1, sizeof( char));
    tseq = calloc( l1+l2+1, sizeof( char));

    i  = l1 - 1;
    j  = l2 - 1;
    i1 = l1 - 1;
    i2 = l2 - 1;

    do {
      if( verbose )
      printf(     "%d %d\n", i, j);

      if( path[j*l1+i] == 0) {
       if( i1 >= 0 ) {
       qseq[is] = seq1[i1];
       i1--;
       }
       else {
       qseq[is] = '-';
       mseq[is] = ' ';
       }
       if( i2 >= 0 ) {
       tseq[is] = seq2[i2];
       i2--;
       }
       else {
```

Fig. 17 (contd.)

```
            tseq[is] = '-';
            mseq[is] = ' ';
            }
            /*
2695        if( tseq[is] == qseq[is]     )   { mseq[is] =
    ':'; }
            else                  { mseq[is] = ' '; }
            */
            mseq[is]   = match_sym( tseq[is], qseq[is], pa-
2700  ram->score_rec);
            is++;
            i--;
            j--;
            if( i<0 && j<0 ) {
2705        done =     1;
            }
           }
           else if( path[j*l1+i] == 2) {
            qseq[is]   = '-';
2710        mseq[is]   = ' ';
            tseq[is]   = seq2[i2];
            i2--;
            is++;
            j--;
2715       }
           else if( path[j*l1+i] == 1) {
            qseq[is]   = seq1[i1];
            i1--;
            mseq[is]   = ' ';
2720        tseq[is]   = '-';
            is++;
            i--;
           }
           if( i < 0   ) i = 0;
2725       if( j < 0   ) j = 0;
           if( verbose )
           printf(     "%d %d %c %c %c\n", i, j, qseq[is-1],
    mseq[is-1], tseq[is-1]);
       } while( !done);
2730   qseq[is] = 0;
       mseq[is] = 0;
       tseq[is] = 0;

       fprintf( outfile, "%s\n", reverse( qseq));
2735   fprintf( outfile, "%s\n", reverse( mseq));
       fprintf( outfile, "%s\n", reverse( tseq));

       free( qseq);
       free( mseq);
```

Fig. 17 (contd.)

```
2740        free( tseq);
          }
        }


2745    int align_local_traceback( FILE *outfile, param_type
        *param,
            int l1, int l2, int *idx_seq1, int *idx_seq2,
            char *mask_seq1, char *mask_seq2, char *seq1, char
        *seq2,
2750        int *mat, int *path){

          stack_type      *stack;
          int       i, j, k;
          pair_type pair;
2755      char            *annot;
          matrix_type     *mt;
          int       score, maxscore, firstscore=0, thes-
        core=INT_MAX, max_i, max_j;
          int       n_hyp = 0;
2760      char            *outstr[3];
          char            *qseq, *mseq, *tseq;

          stack           = init_stack( STACK_CHUNK);

2765      mt        = calloc( 1, sizeof( matrix_type));
          mt->d           = mat;
          mt->l1    = l1;
          mt->l2    = l2;

2770      qseq = calloc( l1+l2+1, sizeof( char));
          mseq = calloc( l1+l2+1, sizeof( char));
          tseq = calloc( l1+l2+1, sizeof( char));

          while( thescore >= param->min_score && n_hyp < param-
2775    >max_res) {

            maxscore = INT_MIN;
            for( j=0; j<l2; j++) {
              for( i=0; i<l1; i++) {
2780            score = mat[ matidx( i, j, l1, l2)];
                if(     score >     maxscore) {
                  maxscore = score;
                  max_i  = i;
                  max_j  = j;
2785            }
              }
            }
```

Fig. 17 (contd.)

```
        if( ! firstscore && maxscore >= param->min_score)
    firstscore = maxscore;
2790
        push( stack, setpair( max_i, max_j));
        thescore = mat[ matidx( max_i, max_j, l1, l2)];
        mat[  matidx( max_i, max_j, l1, l2)] = 1;
        /* Trace back */
2795
        k=0;
        while( stack->sp > 0 ) {
          pair = pop( stack);
          if(0){ printf( "pop        :" ); print_pair(
2800  pair);}
          i     = pair.i;
          j     = pair.j;
          if( path[ matidx( i, j, l1, l2)] == 1 ) {
          if( i>0 && mat[ matidx( i-1, j, l1, l2)] > 0 ) {
2805        push( stack, setpair( i-1, j));
          }
            qseq[k] = seq1[i];
            mseq[k] = ' ';
            tseq[k] = '-';
2810        k++;
          }
          else if( path[ matidx( i, j, l1, l2)] == 2 ) {
          if( j>0 && mat[ matidx( i, j-1, l1, l2)] > 0) {
            push( stack, setpair( i, j-1));
2815      }
            qseq[k] = '-';
            mseq[k] = ' ';
            tseq[k] = seq2[j];
            k++;
2820      }
          else if( path[ matidx( i, j, l1, l2)] <= 0 ) {
          if(  i>0 && j>0 && mat[ matidx( i-1, j-1, l1,
    l2)] > 0    ) {
            push( stack, setpair( i-1, j-1));
2825      }

            qseq[k] = seq1[i];
            tseq[k] = seq2[j];
            mseq[k] = match_sym( tseq[k], qseq[k], param-
2830  >score_rec);
            k++;
          if(0)printf( " ( ) %d %d\n", j,     i );
          }
        }
2835    if(     thescore >= param->min_score ) {
          n_hyp++;
```

Fig. 17 (contd.)

```
            if( n_hyp   <= param->max_res) {
            qseq[k]    = 0;
            mseq[k]    = 0;
2840        tseq[k]    = 0;
            outstr[0] = reverse( qseq);
            outstr[1] = reverse( mseq);
            outstr[2] = reverse( tseq);
              if( outfile != NULL ) {
2845          fprintf( outfile, "Score= %d\n", thescore);
              print_lines( outfile, outstr, 3);
              }
            }
          }
2850      annot          = empty_str( l1);
          free( annot);
        }

        free( mt);
2855    free( qseq);
        free( mseq);
        free( tseq);
        free_stack( stack);
        return firstscore;
2860  }


      void out_align_fill( param_type *param, int l1, int l2,
      int *idx_seq1,
2865      int *idx_seq2, int *mat, int *path, char *seq1,
      char *seq2){
        int  i, j, k,   max_path;
        int  score, max_score;
        int  h_size =   3;
2870    int  gap_score;
        int  match_factor;

        /*
           0 = match or mismatch, go diagnoally
2875       1 = insertion in query, go    horizontally
           2 = gap in   query, go vertically
         */

        for( j=0; j<l2; j++) {
2880      for( i=0; i<l1; i++) {
            if(1)printf( "%d %d\n", i, j);

            if( path[   matidx(     i, j, l1, l2)] == -1 )
            gap_score =  param->score_rec->gap_cont;
2885        else
```

Fig. 17 (contd.)

```
        gap_score =  param->score_rec->gap_start;
        max_score   =  mat[     matidx(     i, j, l1, l2)]
- gap_score;
        max_path    =  -1;
2890
        if( path[   matidx(     i, j+1,     l1, l2)] == -2
)
        gap_score =  param->score_rec->gap_cont;
        else
2895    gap_score =  param->score_rec->gap_start;
        score =  mat[     matidx(     i, j+1,     l1, l2)]
- gap_score;
        if( score   > max_score) { max_score = score;
max_path = -2;}
2900
        if( i-j <= h_size ) {
        score = param->score_rec->loop_score;
        }
        else {
2905    if( path[ matidx( i-1, j+1, l1, l2)] == 0 )
          match_factor =  1;
        else
          match_factor =  param->score_rec-
>match_cont_factor;
2910    score =  mat[     matidx(     i-1, j+1, l1, l2)] +
          match_factor *
          param->score_rec->mat[ idx_seq1[i]      *
          param->score_rec->alph_len + idx_seq2[j]];
        }
2915
        if(0) {
        printf( "%d %d %d %d    %d %d %d\n", i,   j,
idx_seq1[i], idx_seq2[j],
            param->score_rec->alph_len,
2920        param->score_rec->mat[ idx_seq1[i] * param-
>score_rec->alph_len +
            idx_seq2[j]],
            idx_seq1[i] * param->score_rec->alph_len +
idx_seq2[j]);
2925    }

        if( score   > max_score) { max_score = score;
max_path = 0;}
        for( k=j+1; k<i; k++) {
2930    score =      mat[ matidx( i, k+1, l1, l2)] + mat[
matidx( k, j, l1, l2)]
              - param->score_rec->gap_start;
        if( score > max_score) { max_score = score;
max_path = k;}
```

Fig. 17 (contd.)

```
2935        if(0)printf( "%d %d  %d %d  %d\n", i, k, k, j,
      score);
            }

            if( max_score > 0 ) {
2940        mat[ matidx( i, j, l1, l2)] = max_score;
            }
            else {
            mat[ matidx( i, j, l1, l2)] = 0;
            }
2945        path[ matidx( i, j, l1, l2)] = max_path;
            if(0) {
            printf("%c%c      %3d %3d     score: %3d  path:
      %3d\n", seq1[i], seq2[j],
                i, j, max_score, max_path);
2950        }
          }
        }
      }

2955
      void align_fill( param_type *param, int l1, int l2, int
      *idx_seq1,
          int *idx_seq2, int *mat, int *path, char *seq1,
      char *seq2){
2960    int  i, j;
        char c1[2], c2[2];
        int  i1, i2;
        int  sm, si, sd, max_s;
        int  dir;
2965    int  match_score;

        if( verbose > 3 )
          printf( "%d   %d\n", l1, l2);

2970    /*
           -1 = mismatch, go diagnoally
            0 = match, go diagnoally
            1 = insertion in query, go horizontally
            2 = gap in query, go vertically
2975    */

        for( j=0; j<l2; j++) {
          c2[0] = seq2[j];

2980      c2[1] = 0;
          i2 = strcspn( param->score_rec->alph, c2);
          for( i=0; i<l1; i++) {
            c1[0] = seq1[i];
```

Fig. 17 (contd.)

```
            c1[1] = 0;
2985        i1    = strcspn( param->score_rec->alph, c1);


            if( i > 0   && j > 0) {
            sm    =     mat[(j-1)*l1+(i-1)];
2990
            }
            else {
             /*
            if( i > 0 ) {
              sm    = param->score_rec->gap_start +
                (i-1) * param->score_rec->gap_cont;
            }
            else if( j > 0 ) {
              sm    = param->score_rec->gap_start +
3000            (j-1) * param->score_rec->gap_cont;
            }
              */
            sm = 0;
            }
3005        match_score = param->score_rec->mat[ i2*param-
      >score_rec->alph_len+i1];
            dir = 0;
            if( param->score_rec->mismatch_cont < 0 )
              sm     += match_score;
3010        else {
             if( match_score < 0 ) {
              dir    = -1;
              if( (i > 0) && (j > 0) && (path[(j-1)*l1+(i-1)]
      != 0)) {
3015            sm    -= param->score_rec->mismatch_cont;
              }
              else {
                sm     += match_score;
              }
3020         }
             else {
               sm     += match_score;
              }
            }
3025        /* Add mismatch continuation score here */

            /*if( (i > 0) && (path[j*l1+(i-1)] > 0)) { */
            if( i > 0   ) {
            si    =     mat[j*l1+(i-1)];
3030        }
            else {
            si    =     0;
```

Fig. 17 (contd.)

```
            }

3035        /*if( (j > 0) && (path[(j-1)*l1+i] > 0)) { */
            if( j > 0   ) {
            sd    =     mat[(j-1)*l1+i];
            }
            else {
3040        sd    =     0;
            }

            if( i>0 && path[j*l1+(i-1)] > 0) {
            si   -=     param->score_rec->gap_cont;
3045        }
            else {
            si   -=     param->score_rec->gap_start;
            }

3050        if( j>0 && path[(j-1)*l1+i] > 0) {
            sd   -=     param->score_rec->gap_cont;
            }
            else {
            sd   -=     param->score_rec->gap_start;
3055        }

            max_s   =   sm;
            if( si > max_s) {
            max_s =     si;
3060        dir   =     1;
            }
            if( sd > max_s) {
            max_s =     sd;
            dir   =     2;
3065        }

            mat[j*l1+i]  = MAX( max_s, 0);
            path[j*l1+i] = dir;
            /*printf(   "%s %s %d %d %d   %d\n", c1, c2, i1,
3070    i2, score, mat[j*l1+i]);*/
          }
        }

        if( 0)    {
3075      print_mat( seq1, seq2, mat);
          printf( "\n");
          print_mat( seq1, seq2, path);
          printf( "\n");
        }
3080
        /*  printf( "score= %d\n", mat[l1*l2-1]); */
```

Fig. 17 (contd.)

```
}

int align( FILE *outfile, param_type *param, char
*seq1, char *seq2){
  int  l1, l2;
  int  *mat, *path;

  char *mask_seq1, *mask_seq2;
  int  *idx_seq1, *idx_seq2;
  int  score       = INT_MAX;

  l1      = strlen( seq1);
  l2      = strlen( seq2);
  mat     = calloc( matsize( l1, l2), sizeof( int));
  path          = calloc( matsize( l1, l2), sizeof(
int));
  mask_seq1 = calloc( l1 + 1, sizeof( char));
  mask_seq2 = calloc( l2 + 1, sizeof( char));
  strcpy( mask_seq1, seq1);
  strcpy( mask_seq2, seq2);


  if(0)     {
    print_mat( seq1, seq2, mat);
    printf( "\n");
    if(1) print_mat( seq1, seq2, path);
    printf( "\n");
  }

  align_fill( param, l1, l2, idx_seq1, idx_seq2, mat,
path,
      mask_seq1, mask_seq2);
  if(0)     {
    print_mat( seq1, seq2, mat);
    printf( "\n");
    if(1) print_mat( seq1, seq2, path);
    printf( "\n");
  }

  score = align_local_traceback( outfile, param, l1,
l2, idx_seq1, idx_seq2,
      mask_seq1, mask_seq2, seq1, seq2, mat, path);
  free( mat);
  free( path);
  free( mask_seq1);
  free( mask_seq2);
  return score;
}
```

Fig. 17 (contd.)

```
/******* Nussinov - secondary structure  prediction
******/
3135

int nussinov_local_traceback( FILE *outfile, param_type
      *param,     int depth,
                  int l, int* idx_seq, char *seq,
3140              int *mat, int *path){
  stack_type      *stack;
  int       i, j, k;
  pair_type pair;
  char            *annot;
3145  matrix_type     *mt;
  int       score, maxscore, max_i, max_j;
  int       maxpath;
  int       n_hyp = 0;
  char            *outstr[2];
3150
  stack           = init_stack( STACK_CHUNK);

  mt        = calloc( 1, sizeof( matrix_type));
  mt->d           = mat;
3155  mt->l1    = l;
  mt->l2    = l;

  annot           = empty_str( l);

3160    maxscore = INT_MIN;
    for( j=0; j<l; j++) {
      for( i=j; i<depth + j + 1 && i<l; i++) {
      score = mat[ matidx( i, j, l, l)];
      if( score > maxscore) {
3165    maxscore = score;
        max_i    = i;
        max_j        = j;
      }
      }
3170  }

    mat[ matidx( max_i, max_j, l, l)] = 1;

    push( stack, setpair( max_i, max_j));
3175
    /* Trace back */

    while( stack->sp > 0 ) {
      pair = pop( stack);
```

Fig. 17 (contd.)

```
3180        if(0){ printf( "pop       :" ); print_pair(
      pair);}
            i     = pair.i;
            j     = pair.j;
            if( i > j   ) {
3185        if( path[ matidx( i, j, l, l)] == -1 ) {
              if( mat[ matidx( i-1, j, l, l)] > 0) {
                push( stack, setpair( i-1, j));
              }
            }
3190        else if( path[ matidx( i, j, l,     l)] == -2 ) {
              if( mat[ matidx( i, j+1, l, l)] > 0) {
                push( stack, setpair( i, j+1));
              }
            }
3195        else if( path[ matidx( i, j, l,     l)] == 0 ) {
              if( mat[ matidx( i-1, j+1, l,     l)] > 0     )
      {
                push( stack, setpair( i-1, j+1));
              }
3200          if( param->score_rec->match_threshold  <=
                   param->score_rec->mat[ idx_seq[i] * param-
      >score_rec->alph_len +  idx_seq[j]]) {
                annot[i] = ')';
                annot[j] = '(';
3205          }
              else {
                annot[i] = '.';
                annot[j] = '.';
              }
3210          if(0)printf( " ( ) %d %d\n", j,   i );
            }
            else {
              k = path[ matidx( i, j, l, l)];
              push(     stack, setpair(   k , j));
3215          push(     stack, setpair(   i , k+1));
            }
            }
          }

3220      if(     maxscore >= param->min_score ) {
            n_hyp++;
            if( n_hyp   <= param->max_res) {
            if(0)printf( "%s\n", seq );
            if(0)printf( "%s\n", annot );
3225        outstr[0] = seq;
            outstr[1] = annot;
              if( outfile != NULL ) {
              fprintf( outfile, "Score= %d\n", maxscore);
```

Fig. 17 (contd.)

```
            print_lines( outfile, outstr, 2);
3230        }
          }
        }
        free( annot);
        annot         = empty_str( l);
3235

      free( mt);
      free( annot);
      free_stack( stack);
3240
      return maxscore;
    }


3245 char *nussinov_traceback( FILE *outfile,  param_type
    *param,
                          int depth, int l, int* idx_seq,
                          char *seq, int *mat, int *path){
      stack_type      *stack;
3250  int       i, j, k, d;
      pair_type pair;
      char            *annot;
      matrix_type     *mt;
      int       gap_score1, gap_score2;
3255  char            *outstr[2];

      stack           = init_stack( STACK_CHUNK);

      mt        = calloc( 1, sizeof( matrix_type));
3260  mt->d           = mat;
      mt->l1    = l;
      mt->l2    = l;

      annot           = empty_str( l);
3265
      for( d=depth;   d<l; d++) {

        push( stack, setpair( d, d-depth));

3270    while( stack->sp > 0 ) {
          pair = pop( stack);
          if(0){ printf( "pop        :" ); print_pair(
    pair);}
          i     = pair.i;
3275      j     = pair.j;
          if( i > j   ) {
          if( path[ matidx( i, j, l, l)] == -1 ) {
```

Fig. 17 (contd.)

```
            push(     stack, setpair(   i-1, j));
          }
3280      else if( path[ matidx( i, j, l,     l)] == -2 ) {
            push(     stack, setpair(   i, j+1));
          }
          else if( path[ matidx( i, j, l,     l)] == 0 ) {
            if( param->score_rec->match_threshold   <=
3285            param->score_rec->mat[ idx_seq[i]   * param-
>score_rec->alph_len + idx_seq[j]]) {
              annot[i] = ')';
              annot[j] = '(';
            }
3290        else {
              annot[i] = '.';
              annot[j] = '.';
            }
            if(0)printf( " ( ) %d %d\n", j,   i );
3295        push(     stack, setpair(   i-1, j+1));
          }
          else {
                k     = path[     matidx(     i, j, l, l)];
              push( stack, setpair( k ,     j));
3300          push( stack, setpair( i ,     k+1));
          }
      if(0){
          if( path[ matidx( i-1, j, l, l)] == -1 )
            gap_score1 =    param->score_rec->gap_cont;
3305      else
            gap_score1 =    param->score_rec->gap_start;
          if( path[ matidx( i, j+1, l, l)] == -2 )
            gap_score2 =    param->score_rec->gap_cont;
          else
3310        gap_score2 =    param->score_rec->gap_start;

          if(     m( i-1,   j,    mt) ==
                  m( i,         j,      mt) - gap_score1) {
            if(0)printf( " mat: -1 %d %d\n", m( i-1, j,
3315  mt), m( i, j, mt));
            push(     stack, setpair(   i-1, j));
          }
          else if(  m( i,       j+1, mt) ==
                  m( i,         j,       mt) - gap_score2) {
3320        if(0)printf( " mat: -2 %d %d\n", m( i-1, j,
      mt), m( i, j, mt));
            push(     stack, setpair(   i, j+1));
          }
          else if( (mat[ matidx( i-1,   j+1, l, l)] +
3325              param->score_rec->mat[ idx_seq[i] * pa-
      ram->score_rec->alph_len
```

Fig. 17 (contd.)

```
                    + idx_seq[j]])
                ==
                 mat[ matidx( i,     j,   l, l)]) {
3330          if(0)printf( " mat:  0 %d %d\n", m( i-1, j,
    mt), m( i, j, mt));
              annot[i] = ')';
              annot[j] = '(';
              if(0)printf( " ( ) %d %d\n", j,   i );
3335          push(     stack, setpair(   i-1 , j+1));
            }
            else {
              for( k=j+1; k<i; k++) {
                if(0)printf( "  %d %d   %d %d\n", i, k, k,
3340  j);
                if(     mat[ matidx( i,   k+1, l,     l)] +
    mat[ matidx( k, j, l, l)]     ==
                   mat[ matidx( i,     j,   l, l)]) {
                   if(0)printf( " mat:  %d %d %d\n",   k, m( i-
3345  1, j, mt), m( i, j, mt));
                   push( stack, setpair( k ,     j));
                   push( stack, setpair( i ,     k+1));
                   break;
                }
3350          }
            }
    }
            }
          }
3355
          if( outfile != NULL ) {
            fprintf( outfile, "Score= %d start= %d\n",
            mat[ matidx( d, d-depth, l, l)], d - depth + 1);
            outstr[0] = seq;
3360        outstr[1] = annot;
            print_lines( outfile, outstr, 2);
          }
          free( annot);
          annot         = empty_str( l);
3365
        }

        free_stack( stack);
        return annot;
3370  }


    void nussinov_fill( param_type *param, int depth, int
    l, int *idx_seq,
3375                    int *mat, int *path, char *seq){
```

Fig. 17 (contd.)

```
        int  d;
        int  i, j, k,   max_path;
        int  score, max_score;
        int  h_size =   3;
3380    int  gap_score;
        int  match_factor;

        for( d=0; d<depth; d++) {
          for( i=d+1;   i<l; i++) {
3385        if(0)printf( "%d %d %d\n", d, i, depth);
            j           = i - 1     - d;

            if( path[   matidx(     i-1, j,     l, l)] == -1 )
            gap_score =  param->score_rec->gap_cont;
3390        else
            gap_score =  param->score_rec->gap_start;

            max_score   =  mat[     matidx(     i-1, j,     l,
    l)] - gap_score;
3395        max_path    =  -1;


            if( path[   matidx(     i, j+1,     l, l)] == -2 )
            gap_score =  param->score_rec->gap_cont;
3400        else
            gap_score =  param->score_rec->gap_start;

            score =  mat[     matidx(     i, j+1,     l, l)] -
    gap_score;
3405        if( score   > max_score) {
              max_score = score;
            max_path  = -2;
            }

3410
            if( i-j <= h_size ) {
            score = param->score_rec->loop_score;
            }
            else {
3415        if( path[ matidx( i-1, j+1, l, l)] == 0   )
              match_factor =  1;
            else
              match_factor =  param->score_rec-
    >match_cont_factor;
3420        score =  mat[     matidx(     i-1, j+1, l, l)] +
              match_factor *
              param->score_rec->mat[ idx_seq[i] * param-
    >score_rec->alph_len +
              idx_seq[j]];
```

Fig. 17 (contd.)

```
3425        }
            if(0)printf( "%d %d %d %d    %d %d %d\n", i,   j,
      idx_seq[i], idx_seq[j],
              param->score_rec->alph_len,
              param->score_rec->mat[ idx_seq[i] * param-
3430  >score_rec->alph_len +
              idx_seq[j]], idx_seq[i] * param->score_rec-
      >alph_len + idx_seq[j]);

            if( score   > max_score) {
3435        max_score = score;
            max_path  = 0;
            }

            for( k=j+1; k<i; k++) {
3440          if( path[ matidx( i, k+1, l, l)] == -2 ||
                path[ matidx( k, j, l, l)] == -1 )
              gap_score =  param->score_rec->gap_cont;
            else
              gap_score =  param->score_rec->gap_start;
3445
            score =      mat[ matidx( i, k+1, l, l)] + mat[
      matidx( k, j, l, l)]
              - gap_score;

3450        if( score > max_score) {
              max_score = score;
              max_path  = k;
            }
            if(0)printf( "%d %d  %d %d  %d   (%d, %d)\n", i,
3455  k, k, j, score,
                  mat[ matidx( i, k+1, l, l)],
                  mat[ matidx( k, j, l, l)]);
            }

3460        if( max_score > 0 ) {
            mat[  matidx( i, j, l, l)] = max_score;
            }
            else {
            mat[  matidx( i, j, l, l)] = 0;
3465        }
            path[ matidx( i, j, l, l)] = max_path;
            if(0)printf("%c%c %3d %3d     score: %3d  path:
      %3d\n", seq[i], seq[j], i, j, max_score, max_path);
          }
3470    }
      }
```

Fig. 17 (contd.)

```
int nussinov( FILE *outfile, param_type *param, char
*seq){
  int  l, i;
  int  *mat, *path;
  int  *idx_seq;
  int  score = INT_MAX;
  int  max_score = 0;
  int  depth  =   param->depth;


  l      = strlen( seq);
  if( depth > l-1 ) { depth = l-1;}
  mat    = calloc( matsize( l, l), sizeof( int));
  path         = calloc( matsize( l, l), sizeof( int));


  idx_seq = seq2idx_seq( seq, param->score_rec);
  nussinov_fill( param, depth, l, idx_seq, mat, path,
seq);
  if( 0)    {
    print_mat( seq, seq, mat);
    printf( "\n");
    print_mat( seq, seq, path);
    printf( "\n");
  }

  if( param->global) {
    nussinov_traceback( outfile, param,   depth, l,
idx_seq, seq,     mat, path);
  }
  else {
    i =     0;
    while( score > param->min_score && i < param-
>max_res) {
      score = nussinov_local_traceback(  outfile, pa-
ram,  depth, l,
                        idx_seq, seq, mat, path);
      i++;
      if( score   > max_score) max_score = score;

    }
  }


  free(     mat);
  free(     path);
  free(     idx_seq);
  return max_score;
}
```

Fig. 17 (contd.)

```
/********* OLIGO ******************/

void  calc_olig( int id, char *seq, char *rev_seq,
       int *self, int *target, param_type *param, int
p){
  int score_self, score_target, score_struc;

    strcpy( rev_seq, seq);
    reverse( rev_seq);
    score_self = align( NULL, param, seq, rev_seq);
    lower( rev_seq);
    comp( rev_seq);
    reverse( rev_seq);
    score_target = align( NULL, param, seq, rev_seq);
    if( p) {
     if(0) {
      score_struc  = nussinov( NULL, param, seq);
      printf( "%7d %s %7d %7d %7d\n", id, seq,
score_self, score_target,
                  score_struc);
     }
     else {
      if( score_target > score_self + 5) {
        printf( "%d %s %d %d\n", id, seq, score_self,
score_target);
      }
     }
    }
    *self   = score_self;
    *target = score_target;
}


long spikenum( int n){
  int i;
  long n_tmp;
  long mask = 1;
  long res = 0;
  int  nbit = 14; /* 16384 different spikings in 32
bits */

/*
   To get all spike variants of a given
*/

  for( i=0;i<nbit;i++){
```

Fig. 17 (contd.)

```
    n_tmp  = n & mask;
    n_tmp  = n_tmp << (i+i+2);
    res   += n_tmp;
    mask   = mask << 1;
  }

  return res;
}

long lna2spikenum( long n){
  int i;
  long n_tmp;
  long mask = 4;
  long res = 0;
  int  nbit = 14; /* 16384 different spikings in 32
bits */

/*
   To get all spike variants of a given
*/

  for( i=0;i<nbit;i++){
    n_tmp  = n & mask;
    n_tmp  = n_tmp >> (i+i+2);
    res   += n_tmp;
    mask   = mask << 3;
  }

  return res;
}


long reverse_dna( long n, int oligolen){
  int        i;
  int        nbit = oligolen * 2;
  long       mask = 3;
  long       n_tmp;
  long       res = 0;

  for( i=0;i<nbit/2;i++){
    n_tmp  = n & mask;
    n_tmp  = n_tmp >> i*2;
    n_tmp  = n_tmp << (nbit - (i+1)*2);
    res   += n_tmp;
    mask   = mask << 2;
  }
  return res;
}
```

Fig. 17 (contd.)

```
long comp_dna( long n, int oligolen) {
  long       res;
  long       mask = (n_nmer( oligolen, "acgt") - 1);

  res = ~n & mask;
  return res;
}

long rev_comp_dna( long n, int oligolen){
  long       res;
  res = reverse_dna( n, oligolen);
  res = comp_dna( res, oligolen);
}


long dna2lna( int n){
  int i;
  long n_tmp;
  long mask = 3;
  long res = 0;
  const int  nbit = 32;

/*
   To get all spike variants of a given
*/

  for( i=0;i<nbit/2;i++){
    n_tmp  = n & mask;
    n_tmp  = n_tmp << i;
    res   += n_tmp;
    mask   = mask << 2;
  }

  return res;
}


long lna2dna( int n){
  int i;
  long n_tmp;
  long mask = 3;
  long res = 0;
  const int  nbit = 32;

/*
   To get the dna only sequence
```

Fig. 17 (contd.)

```
3670   */

         for( i=0;i<nbit/2;i++){
           n_tmp  = n & mask;
           n_tmp  = n_tmp >> i;
3675       res   += n_tmp;
           mask   = mask << 3;
         }

         return res;
3680   }


       long dna2lna_spike( int nmer, int spike_pat) {
         long res;
3685     res  = dna2lna( nmer);
         res += spikenum( spike_pat);
         return res;
       }

3690


       void allolig( FILE *outfile, int oligolen, int
       oligosample, int spikefreq,
           char *n_mer_range, param_type *param){
3695     char  alph[255] = "acgtACGT";
         long  n, nmer;
         char  *seq, *rev_seq;
         long  i, j;
         long  n0 = 0, n1;
3700     int   spike_0 = 0, spike_1, spike_n;
         int   dna_flag = 0;
         int   self_s, target_s;

         check_endian();
3705     check_limits();

         if( n_mer_range != NULL ) {
           if( sscanf( n_mer_range, "%d-%d:%d-%d", &n0, &n1,
       &spike_0, &spike_1) == 4){
3710         dna_flag = 1;
             n        = n1 + 1;
             spike_n  = spike_1 - spike_0 + 1;
           }
           else if( sscanf( n_mer_range, "%d-%d", &n0, &n1)
3715   == 2){
             n       = n1 + 1;
             spike_n = n_nmer( oligolen, "aA");
           }
```

Fig. 17 (contd.)

```
      else {
3720    die( "Failed to parse %s, expected %%d-%%d or
%%d-%%d:%%d-%%d",
            n_mer_range);
      }
    }
3725  else {
      n         = n_nmer( oligolen, alph);
      spike_n   = n_nmer( oligolen, "aA");
    }

3730  rev_seq   = empty_str( oligolen);

    param->min_score = 2;
    param->max_res   = 1;

3735  if( oligosample ) {
      for( j=0; j<oligosample; j++) {
        seq = make_random_nmer( oligolen, spikefreq,
alph);
        calc_olig( j, seq, rev_seq, &self_s, &target_s,
3740 param, 1);
        free( seq);
      }
    }
    else {
3745    if( dna_flag) {
        for( i=n0; i<n; i++) {
          for( j=spike_0; j<spike_n; j++) {
            nmer = dna2lna_spike( i, j);
            seq  = make_nmer( nmer, oligolen, alph);
3750        calc_olig( nmer, seq, rev_seq, &self_s,
&target_s, param, 1);
            free( seq);
          }
        }
3755    }
      else {
        for( j=n0; j<n; j++) {
          seq = make_nmer( j, oligolen, alph);
          calc_olig( j, seq, rev_seq, &self_s, &target_s,
3760 param, 1);
          free( seq);
        }
      }
    }
3765  free( rev_seq);
}
```

Fig. 17 (contd.)

```
      /************ CLUSTER *****************/
3770

      int cluster( FILE *outfile, int score_cutoff, int
      n_okseq, param_type *param){
        int i;
3775    int *seq_id2cluster, *cluster2seq_id;
        int seq_id = 0, next_seq_id = 0, cluster_id = 0;
        int score;
        int nseq;
        int seq_id_1;
3780    sequences_type *seqs;

        param->max_res = 1;
        param->min_score = 1;

3785
        nseq             = param->seqs->nseq;
        seq_id2cluster = salloc( nseq, sizeof( int));
        cluster2seq_id = salloc( nseq, sizeof( int));

3790    for( i=0; i< nseq; i++)
          seq_id2cluster[i] = -1;

        do{
          seq_id                  = next_seq_id;
3795      cluster2seq_id[ cluster_id]  = seq_id;
          seq_id2cluster[ seq_id]      = cluster_id;
          seq_id_1 = seq_id + 1;

          if( seq_id_1 < n_okseq )
3800        next_seq_id = seq_id_1;
          else
            next_seq_id             = 0;

      printf( "%d\n", seq_id);
3805
          for( i=seq_id_1; i<nseq; i++){
            if( seq_id2cluster[ i] < 0 ) {

            score = align( NULL, param,
3810            param->seqs->seqs[ seq_id], param->seqs-
      >seqs[i]);

              if( score >= score_cutoff) {
                seq_id2cluster[ i] = cluster_id;
3815          }
              else if( next_seq_id == 0) {
```

Fig. 17 (contd.)

```
              next_seq_id = i;
            }

3820        }
          }

          cluster_id++;
        } while ( next_seq_id > 0);
3825

        seqs = new_sequences_type( SEQS_CHUNK);
        for( i=0; i<cluster_id; i++) {
          add_seq2seqs( param->seqs->seqs[i], seqs);
3830    }
        print_fasta( outfile, seqs);
        free_sequences_type( seqs);


3835    free( cluster2seq_id);
        free( seq_id2cluster);
        return cluster_id;
      }

3840
      /******* ARGUMENT PARSING AND INITIALIZATION
      **************/


3845  param_type *init(){
        char            a[255];
        int  i, j;
        param_type *param;
        int  *mat;
3850  #define     ALPHLEN     9
        int  mathyp_init[ALPHLEN][ALPHLEN]= {
                          { -3, -3, -3,  3, -5, -5, -5,  5, -
      99},
                          { -3, -3,  5, -3, -5, -5,  9, -5, -
3855  99},
                          { -3,  5, -3,  1, -5,  9, -5,  2, -
      99},
                          {  3, -3,  1, -3,  5, -5,  2, -5, -
      99},
3860                      { -5, -5, -5,  5, -8, -8, -8,  8, -
      99},
                          { -5, -5,  9, -5, -8, -8, 14, -8, -
      99},
                          { -5,  9, -5,  2, -8, 14, -8,  4, -
3865  99},
```

Fig. 17 (contd.)

```
                        {  5, -5,  2, -5,  8, -8,  4, -8, -
    99},
                        {-99,-99,-99,-99,-99,-99,-99,-99, -
    99}
                       };
    /*
      int  mathyp_init[ALPHLEN][ALPHLEN]= {
                              { -2, -2, -2,  3, -2, -2, -2,
    5, -99},
                              { -2, -2,  5, -2, -2, -2,  9,
    -2, -99},
                              { -2,  5, -2,  1, -2,  9, -2,
    2, -99},
                              {  3, -2,  1, -2,  5, -2,  2,
    -2, -99},
                              { -2, -2, -2,  5, -2, -2, -2,
    8, -99},
                              { -2, -2,  9, -2, -2, -2, 14,
    -2, -99},
                              { -2,  9, -2,  2, -2, 14, -2,
    4, -99},
                              {  5, -2,  2, -2,  8, -2,  4,
    -2, -99},
                       {-99,-99,-99,-99,-99,-99,-99,-99, -
    99}
                      };
    */

      param      = calloc( 1, sizeof( param_type));

      /* alph */
      strcpy( a, "acgtACGT-");
      param->alph            = calloc( strlen( a) + 1,
    sizeof( char));
      strcpy( param->alph, a);

      /* seqs */
      param->seqs            = new_sequences_type(
    SEQS_CHUNK);
      param->seqs2           = new_sequences_type(
    SEQS_CHUNK);

      /* hybridization matrix */

      mat = calloc( ALPHLEN * ALPHLEN, sizeof( int));
      for( i=0;i<ALPHLEN;i++){
        for( j=0;j<ALPHLEN;j++){
          mat[i*ALPHLEN+j] = mathyp_init[i][j];
        }
```

Fig. 17 (contd.)

```
3915    }

        /* defaults */
        param->depth       = INT_MAX;
        param->global      = 0;
3920    param->min_score = 20;
        param->max_res     = INT_MAX;


        param->score_rec = init_score_rec_type(
3925        14, 7, 3,
            -40, 2, 1,
            8, 3, 1,
            param->alph, mat);

3930    validate_score_rec_type( param->score_rec, -200,
      200);

        return param;
      }
3935

      #ifndef LIB
      /***************** MAIN *****************/

3940
      int main (int argc, char* argv[]) {
        int         i;
        char              *rev_seq;
        param_type        *param;
3945    int         algo        = 0;

        char              *options;
        int         next_option;
        char*             outfn             = NULL;
3950    char*             infn        = NULL;
        FILE              *outfile, *seqfile;
        int         othermethod       = 0;
        int         oligolen    = 7;
        int         oligosample       = 0;
3955    int         spikefreq   = 0;
        int         score_cutoff      = 100;
        int               n_okseq     = 1;
        char*           n_mer_range     = NULL;

3960    const     char* const   short_options  =
      "hvruayto:i:j:s:z:d:gn:p:m:l:e:f:c:k:w:";
        const     struct option long_options[] = {
          { "help",                 0,      NULL, 'h' },
```

Fig. 17 (contd.)

```
        { "verbose",        0,     NULL, 'v' },
3965    { "version",        0,     NULL, 'r' },
        { "secondary_structure", 0,     NULL, 'u' },
        { "self_anealing",       0,     NULL, 'a' },
        { "hybridization",       0,     NULL, 'y' },
        { "tm",             0,     NULL, 't' },
3970    { "output",              1,     NULL, 'o' },
        { "input",               1,     NULL, 'i' },
        { "input2",              1,     NULL, 'j' },
        { "seq",                 1,     NULL, 's' },
        { "seq2",                1,     NULL, 'z' },
3975    { "depth",               1,     NULL, 'd' },
        { "global",              0,     NULL, 'g' },
        { "min_score",           1,     NULL, 'n' },
        { "max_res",        1,     NULL, 'p' },
        { "matrix",              1,     NULL, 'm' },
3980    { "allolig",        1,     NULL, 'l' },
        { "n_mer_range",         1,     NULL, 'w' },
        { "cluster",        1,     NULL, 'c' },
        { "n_okseq",        1,     NULL, 'k' },
        { "sample",              1,     NULL, 'e' },
3985    { "spikefreq",           1,     NULL, 'f' },
        { NULL,             0,     NULL,  0  }   /* Required
    at end of array.  */
      };

3990  /* mtrace(); debug memory leaks under linux */

      param     = init();

      program_name = argv[0];
3995
      do {
        next_option   = getopt_long (argc, argv,
    short_options,
          long_options, NULL);
4000    switch (next_option)
        {
        case 'h':     /* -h or --help   */
          usage ("");

4005    case 'i':     /* -i or --input */
          infn = optarg;
          seqfile =   sfopen(     infn, "r");
          read_fasta( seqfile, param->seqs,  param->alph);
          fclose( seqfile);
4010      break;

        case 'j':     /* -j or --input2 */
```

Fig. 17 (contd.)

```
            infn = optarg;
            seqfile =   sfopen(     infn, "r");
4015        read_fasta( seqfile, param->seqs2, param->alph);
            fclose( seqfile);
            break;

          case 'o':     /* -o or --output */
4020        outfn = optarg;
            break;

          case 'm':     /* -m or --matrix*/
            options =   optarg;
4025        read_score_rec_type( options, param->score_rec);
            break;

          case 's':     /* -s or --seq*/
            options =   optarg;
4030        add_seq2seqs( options, param->seqs);
            break;

          case 'z':     /* -z or --seq2*/
            options =   optarg;
4035        add_seq2seqs( options, param->seqs2);
            break;

          case 'd':     /* -d or --depth*/
            options =   optarg;
4040        param->depth = atoi( options);
            break;

          case 'g':     /* -d or --depth*/
            param->global = 1;
4045        break;

          case 'p':     /* -p or --max_res*/
            options =   optarg;
            param->max_res = atoi( options);
4050        break;

          case 'n':     /* -n or --min_score*/
            options =   optarg;
            param->min_score = atoi( options);
4055        break;

          case 'v':     /* -v or --verbose */
            verbose += 1;
            break;
4060
          case 'r':     /* -r or --version */
```

Fig. 17 (contd.)

```
            printf( "dyp version %s\n", THEVERSION);
            exit( 0);
            break;
4065
          case 'u':     /* -s or --secondary_structure */
            algo = algo | SECONDARY_STRUCTURE;
            break;

4070      case 'a':     /* -s or --self_anealign */
            algo = algo | SELF_ANEALING;
            break;

          case 'y':     /* -s or --self_anealign */
4075        algo = algo | HYBRIDIZATION;
            break;

          case 't':     /* -t or --tm */
            algo = algo | TM;
4080        break;

          case 'l':     /* -l or --allolig */
            options =   optarg;
            oligolen = atoi( options);
4085        othermethod = ALLOLIG;
            break;

          case 'w':     /* -w or --n_mer_range */
            n_mer_range = optarg;
4090        break;

          case 'c':     /* -c or --cluster */
            options =   optarg;
            score_cutoff = atoi( options);
4095        othermethod = CLUSTER;
            break;

          case 'k':     /* -k or --n_okseq */
            options =   optarg;
4100        n_okseq = atoi( options);
            break;

          case 'e':     /* -p or --sample */
            options =   optarg;
4105        oligosample = atoi( options);
            break;

          case 'f':     /* -f or --spikefreq */
            options =   optarg;
4110        spikefreq = atoi( options);
```

Fig. 17 (contd.)

```
          break;

        case '?':     /* The user specified an invalid op-
      tion.  */
4115      usage ("Sorry, one of the    options    was not
          recogniced");

        case -1:      /* Done     with options.  */
          break;
4120
        default:      /* Something else: unexpected.      */
          abort ();
        }
      }
4125  while     (next_option !=   -1);


      if( outfn == NULL )
        outfile = stdout;
4130  else
        outfile = sfopen( outfn, "w");

      if( verbose)
        print_score_rec_type( param->score_rec);
4135

      if( othermethod == ALLOLIG) {
        allolig( outfile, oligolen, oligosample, spikefreq,
      n_mer_range, param);
4140  }
      else if( othermethod == CLUSTER) {
        cluster( outfile, score_cutoff, n_okseq, param);
      }
      else {
4145    if( !algo )
          usage( "please select a method: u, a or y");

        for( i=0; i<param->seqs->nseq; i++) {
          fprintf( outfile, ">sequence_%03d\n", i+1);
4150      if(   algo & SECONDARY_STRUCTURE )
          nussinov( outfile, param, param->seqs->seqs[i]);
          if( algo & SELF_ANEALING ) {
          rev_seq     = calloc( strlen( param->seqs-
      >seqs[i]) + 1, sizeof( char));
4155      strcpy(     rev_seq, param->seqs->seqs[i]);
          reverse( rev_seq);
          if( param->min_score < 2 )  {
            param->min_score = 2;
          }
```

Fig. 17 (contd.)

```
4160        align( outfile, param, param->seqs->seqs[i],
      rev_seq);
            free( rev_seq);
            }
            if( algo & TM ) {
4165        rev_seq     = calloc( strlen( param->seqs-
      >seqs[i]) + 1, sizeof( char));
            strcpy(     rev_seq, param->seqs->seqs[i]);
              lower( rev_seq);
              comp( rev_seq);
4170        if( param->min_score < 2 )  {
              param->min_score = 2;
            }
            align( outfile, param, param->seqs->seqs[i],
      rev_seq);
4175        free( rev_seq);
            }
            if( algo & HYBRIDIZATION ) {
              if( param->seqs2->seqs[i] == NULL)
                die( "Sequence 2 is missing");
4180        rev_seq     = calloc( strlen( param->seqs2-
      >seqs[i]) + 1, sizeof( char));
            strcpy(     rev_seq, param->seqs2->seqs[i]);
            reverse( rev_seq);
            align( outfile, param, param->seqs->seqs[i],
4185  rev_seq);
            free( rev_seq);
            }
          }
        }
4190    fclose( outfile);

        free_param_type( param);

        return 0;
4195  }
      #endif
```

Fig. 17 (contd.)

SEQUENCE LISTING

<110> Exiqon A/S

<120> PROBES, LIBRARIES AND KITS FOR ANALYSIS OF MIXTURES OF NUCLEIC ACIDS AND METHODS FOR CONSTRUCTING THE SAME

<130> 15769PCT00

<160> 46

<170> PatentIn version 3.2

<210> 1
<211> 23
<212> DNA
<213> artificial sequence

<220>
<223> Synthetic sequence

<400> 1
cgcgtttact ttgaaaaatt ctg 23

<210> 2
<211> 20
<212> DNA
<213> artificial sequence

<220>
<223> Synthetic sequence

<400> 2
gcttccaatt tcctggcatc 20

<210> 3
<211> 25
<212> DNA
<213> artificial sequence

<220>
<223> Synthetic sequence

<400> 3
gcccaagatg ctataaattg gttag 25

<210> 4
<211> 23
<212> DNA
<213> artificial sequence

<220>
<223> Synthetic sequence

<400> 4
gggtttgcaa caccttctag ttc 23

<210> 5
<211> 18
<212> DNA
<213> artificial sequence

<220>
<223> Synthetic sequence

<400> 5
tacggagctg caggtggt 18

<210> 6
<211> 18
<212> DNA
<213> artificial sequence

<220>
<223> Synthetic sequence

<400> 6
gttgggccgt tgtctggt 18

<210> 7
<211> 13
<212> DNA
<213> artificial sequence

<220>
<223> Synthetic sequence

<400> 7
caaggagaag ttg 13

<210> 8
<211> 13
<212> DNA
<213> artificial sequence

<220>
<223> Synthetic sequence

<400> 8
caaggagaag ttg 13

<210> 9
<211> 12
<212> DNA
<213> artificial sequence

<220>
<223> Synthetic sequence

<400> 9

caaggaaagt tg 12

<210> 10
<211> 23
<212> DNA
<213> artificial sequence

<220>
<223> Synthetic sequence

<400> 10
cgcgtttact ttgaaaaatt ctg 23

<210> 11
<211> 20
<212> DNA
<213> artificial sequence

<220>
<223> Synthetic sequence

<400> 11
gcttccaatt tcctggcatc 20

<210> 12
<211> 25
<212> DNA
<213> artificial sequence

<220>
<223> Synthetic sequence

<400> 12
gcccaagatg ctataaattg gttag 25

<210> 13
<211> 23
<212> DNA
<213> artificial sequence

<220>
<223> Synthetic sequence

<400> 13
gggtttgcaa caccttctag ttc 23

<210> 14
<211> 18
<212> DNA
<213> artificial sequence

<220>
<223> Synthetic sequence

<400> 14
tacggagctg caggtggt 18

<210> 15
<211> 18
<212> DNA
<213> artificial sequence

<220>
<223> Synthetic sequence

<400> 15
gttgggccgt tgtctggt 18

<210> 16
<211> 20
<212> DNA
<213> artificial sequence

<220>
<223> Synthetic sequence

<400> 16
gcgagagaaa acaagcaagg 20

<210> 17
<211> 20
<212> DNA
<213> artificial sequence

<220>
<223> Synthetic sequence

<400> 17
attcgtcttc actggcatca 20

<210> 18
<211> 25
<212> DNA
<213> artificial sequence

<220>
<223> Synthetic sequence

<400> 18
cagctaaaaa tgatgacaat aatgg 25

<210> 19
<211> 23
<212> DNA
<213> artificial sequence

<220>
<223> Synthetic sequence

<400> 19
attacatcat gattagggaa tgc 23

<210> 20
<211> 21
<212> DNA
<213> artificial sequence

<220>
<223> Synthetic sequence

<400> 20
gggtttgaac attgatgagg a 21

<210> 21
<211> 20
<212> DNA
<213> artificial sequence

<220>
<223> Synthetic sequence

<400> 21
ggtgtcagct ggaacctctt 20

<210> 22
<211> 13
<212> DNA
<213> artificial sequence

<220>
<223> Synthetic sequence

<220>
<221> modified_base
<222> (1)..(1)
<223> N is LNA methyl cytosine

<220>
<221> misc_feature
<222> (1)..(1)
<223> n is a, c, g, or t

<400> 22
naaggagaag ttg 13

<210> 23
<211> 35
<212> DNA
<213> artificial sequence

<220>
<223> Synthetic sequence

<400> 23
acgtgagctc attgaaactg caggtggtat tatga 35

<210> 24
<211> 44
<212> DNA
<213> artificial sequence

<220>
<223> Synthetic sequence

<400> 24
gatccccggg aattgccatg ctaatcaacc tcttcaaccg ttgg 44

<210> 25
<211> 50
<212> DNA
<213> artificial sequence

<220>
<223> Synthetic sequence

<400> 25
acgtggatcc ttttttttttt ttttttttttt gatccccggg aattgccatg 50

<210> 26
<211> 20
<212> DNA
<213> artificial sequence

<220>
<223> Synthetic sequence

<400> 26
gcgagagaaa acaagcaagg 20

<210> 27
<211> 20
<212> DNA
<213> artificial sequence

<220>
<223> Synthetic sequence

<400> 27
attcgtcttc actggcatca 20

<210> 28
<211> 25
<212> DNA
<213> artificial sequence

<220>

<223> Synthetic sequence

<400> 28
cagctaaaaa tgatgacaat aatgg 25

<210> 29
<211> 23
<212> DNA
<213> artificial sequence

<220>
<223> Synthetic sequence

<400> 29
attacatcat gattagggaa tgc 23

<210> 30
<211> 21
<212> DNA
<213> artificial sequence

<220>
<223> Synthetic sequence

<400> 30
gggtttgaac attgatgagg a 21

<210> 31
<211> 20
<212> DNA
<213> artificial sequence

<220>
<223> Synthetic sequence

<400> 31
ggtgtcagct ggaacctctt 20

<210> 32
<211> 164
<212> DNA
<213> artificial sequence

<220>
<223> Synthetic sequence

<400> 32
caccgttcgg catatccata tttcccacag ccaccaccag gaaggcagca gccaggagga 60

gcagcctcct cagagaagca gcctggagac ttcctccagc tccagggccg ccgcctgctg 120

gagcagcagc accagaagag ggggaggtac ggttggttgt acga 164

<210> 33

<211> 108
<212> DNA
<213> artificial sequence

<220>
<223> Synthetic sequence

<400> 33
tggcggacgc acaccgctta cccctgctgg aggaagctga ggaggagcag cctggagcag 60

cagcagccag ctccgccgcc aggaagccga ctcacgggcc acgcatta 108

<210> 34
<211> 115
<212> DNA
<213> artificial sequence

<220>
<223> Synthetic sequence

<400> 34
gggtgcgacc gtgagtcaat ggtctccagg aggctgtctt ctggtgctgc tcctctgctg 60

cctccagctt ctctggccct ggtggtggct gtgggtaatg cgtggcccgt gagtc 115

<210> 35
<211> 106
<212> DNA
<213> artificial sequence

<220>
<223> Synthetic sequence

<400> 35
attgactcac ggtcgcacca aactctgctg ggctgcctgg aagctccagg agaacttcca 60

gccagctcct ccaccagcag gaagaataac cgtggaacgc ggtcat 106

<210> 36
<211> 124
<212> DNA
<213> artificial sequence

<220>
<223> Synthetic sequence

<400> 36
atacccatcc aaggcgtccc taaaggaggc agaggaaggg agctgccttc ccagcccttc 60

tcccagcaca gcagagcaga gccacctcca gccacatcac caaaatgacc gcgttccacg 120

gtta 124

<210> 37
<211> 115

<212> DNA
<213> artificial sequence

<220>
<223> Synthetic sequence

<400> 37
attgactcac ggtcgcacca aacctggaag gcagaggaac tgcctcctcc accatcacca 60

ctgctgggct gggaagcttc cagcacagca ggaaataacc gtggaacgcg gtcat 115

<210> 38
<211> 121
<212> DNA
<213> artificial sequence

<220>
<223> Synthetic sequence

<400> 38
atacccatcc aaggcgtccc taaacttctc ccagagccac ctccagccag ccacaccagc 60

agagcaggaa ggagctgcct ggagcagctc ccaggagaaa aatgaccgcg ttccacggtt 120

a 121

<210> 39
<211> 115
<212> DNA
<213> artificial sequence

<220>
<223> Synthetic sequence

<400> 39
attgactcac ggtcgcacca aattcctctg ccttcctgct ctgctgggag aaggaggtgg 60

tgatgtggct ggaaggaggc agctccagga gaaaataacc gtggaacgcg gtcat 115

<210> 40
<211> 114
<212> DNA
<213> artificial sequence

<220>
<223> Synthetic sequence

<400> 40
atacccatcc aaggcgtccc taaacttcca ggcagctccc tccagccagc aggacttccc 60

agcccagctc ctccaccagc acagcagagc caaaatgacc gcgttccacg gtta 114

<210> 41
<211> 114
<212> DNA

<213> artificial sequence

<220>
<223> Synthetic sequence

<400> 41
ttagggacgc cttggatggg tatggctgag gcggctggct cctgcatcct cttctgcctc 60

tgctcccagc tgagccatgc cctggcttcc accaattgcc gacccaccgg gata 114

<210> 42
<211> 122
<212> DNA
<213> artificial sequence

<220>
<223> Synthetic sequence

<400> 42
attcgctacg gcccaacacc ttactccacc tcctgcccca ctggggctga agtccagtgt 60

ctggagctgc ttcccagtgg gcagccatcc agcaggccac catatcccgg tgggtcggca 120

at 122

<210> 43
<211> 124
<212> DNA
<213> artificial sequence

<220>
<223> Synthetic sequence

<400> 43
taaggtgttg ggccgtagcg aatcgctctg ccactggggc ctggtctcca tcctctcctc 60

cctgggcaac ctgctgtcct tggcagtggg gaagctgtgc caattgtcct ccgcccggac 120

tcat 124

<210> 44
<211> 122
<212> DNA
<213> artificial sequence

<220>
<223> Synthetic sequence

<400> 44
ttagggacgc cttggatggg tatctctgcc actggctcca gatcctcttc tgccccactg 60

ccatgggcag ctggggcctc ctccctccac ctggcttccc caattgccga cccaccggga 120

ta 122

<210> 45
<211> 118
<212> DNA
<213> artificial sequence

<220>
<223> Synthetic sequence

<400> 45
attcgctacg gcccaacacc ttacctcagc cccagctcca tccagccgcc aaggactggt 60

ctcctgccct gggcaactgg gaatggctgc ttccaccata tcccggtggg tcggcaat 118

<210> 46
<211> 124
<212> DNA
<213> artificial sequence

<220>
<223> Synthetic sequence

<400> 46
taaggtgttg ggccgtagcg aatctgcctc ttcagccgct ctgctcccag ctgagccatc 60

cagtgtgcag gagaggacag caggtggcac agcaggccac caattgtcct ccgcccggac 120

tcat 124