COMPUTER-IMPLEMENTED BIOLOGICAL SEQUENCE IDENTIFIER SYSTEM AND METHOD

This application claims priority to US Provisional Patent Application Nos. 60/691,768, filed on 16 Jun 2005; 60/735,876 filed on 14 Nov 2005; 60/735,824 filed on 14 Nov 2005; and 60/743,639, filed on 22 Mar 2006, and to US Patent Application Nos. 11/422,425, filed on 06 Jun 2006 and 11/422,431, filed on 06 Jun 2006. This application is a continuation-in-part application of US Patent Application Nos. 11/177,647, filed 02 Jul 2005; 11/177,646, filed 02 Jul 2005; and 11/268,373, filed on 07 Nov 2005. These nonprovisional applications claim priority to US Provisional Patent Application Nos. 60/590,931, filed on 02 Jul 2004; 60/609,918, filed on 15 Sep 2004; 60/626,500, filed on 05 Nov 2004; 60/631,437, filed on 29 Nov 2004; and 60/631,460, filed on 29 Nov 2004.

TECHNICAL FIELD

The invention is generally related to processing of biological sequences.

BACKGROUND ART

For both surveillance and diagnostic applications, fine-scale pathogen identification and near-neighbor discrimination is important; therefore, an assay that monitors at this very specific level is desirable in clinical environments (1-3). To successfully use any method based on DNA or RNA detection, these assays must be coupled with large databases of nucleic acid sequence information for assay design to ensure that the desired information is provided and for the interpretation of raw data. Several well established techniques, such as real time PCR, use short, unique stretches of sequenced genomes to provide good specificity (4). These techniques are capable of providing fine-scale identification of several genetically close organisms by selecting a sufficient number of segments. However, these selected segments, which in the initial selection process were specific, are often later found to be less specific as more organisms are sequenced. This is particularly a problem for pathogens belonging to a family with high mutation rate and also for pathogens that have relatively few neighboring pathogens identified. In addition, real time PCR is not capable of detecting the presence of new significant mutations nor can it resolve base sequence details. Similarly, advances in other detection techniques have provided methods of obtaining pathogen identification but suffer from some or all of the problems of using PCR (5-8).

High density resequencing microarrays can produce variable length segments, 10^2-10^5 base pairs (bp), of direct sequence information. They have been successfully used to detect single nucleotide polymorphism (SNP) and genetic variants from viral, bacterial, and eukaryotic genomes (9-16). Their use for SNP detection has clearly established their ability to provide reliable quality sequence information. In most cases, the microarrays were designed to study a limited number of genetically similar target pathogens, and for many cases, the detection methods relied only on recognizing hybridization patterns for identification (12,14,15,17,18). Taking advantage of sequential base resolution capability of resequencing microarrays that was required for SNP detection, resequencing has recently been successfully adapted using a different approach for pathogen identification of multiple bacterial and viral pathogens while allowing for fine detailed discrimination of closely related pathogens and tracking mutations within the targeted pathogen (19-21). The new methodology differed from earlier work by using the resolved bases as the query of a similarity search of DNA databases to identify the most likely species and variants that match the base calls from the hybridization observed. The system was capable of testing for 26 pathogens simultaneously and could detect the presence of multiple pathogens. A software program, REsequencing Pathogen Identifier (REPI), was used to simplify data analysis by performing similarity searches of a genetic database using Basic Local Alignment Search Tool (BLAST) (22). The REPI program used BLAST default settings and would

only return sequences that might represent the hybridization if the expect value, a quantity calculated by the BLAST program that indicates the likelihood that the sequence match found would have occurred by random chance in the database, was less than 10^{-9}. This screened out all cases that had insufficient signal; however, the final determination of what pathogen(s) was detected and to what degree discrimination was possible required manual examination of the returned results. This method successfully allowed fine discrimination of various adenoviruses and strain identifications of Flu A and B samples in agreement with conventional sampling results (19,20). Two important advantages of this approach were that the information was always recovered at the most detailed level possible and that it was capable of still recognizing organisms with recent mutations. This approach also maintained specificity well, as it was not dependent on the uniqueness of short sequences, which are constantly encroached upon as more organisms are sequenced.

Although this analysis method has utility, there are several shortcomings: it is time consuming, not optimized to maximize sensitivity, has complicated results, is suitable only for an expert, and contains redundant or duplicate information. The process was time consuming because only the initial screening was handled automatically while the remaining steps required manual interpretation before the detection analysis was complete. Because a simple criterion (expect value cutoff of 10^{-9}) and non-optimized BLAST parameters were used to consider a pathogen detected, the REPI algorithm provided a list of candidate organisms but did not make a final simple conclusion nor could it relate the results of one prototype sequence to another. Instead, a manual process was used to make the final determination, but because the REPI program provided all similar results and the use of public nucleic acid databases containing redundant entries, a large amount of data was presented to a user that was not useful. In addition, with a manual process it was not possible to establish that the algorithm developed was generally applicable for any organism where nucleic acid base resolved sequence information has been provided.

DISCLOSURE OF THE INVENTION

One method of the invention comprises: submitting a plurality of reference sequences in queries to a taxonomic database to produce a plurality of taxonomic results; and reporting a taxonomic identification based on the taxonomic results. The reference sequences are the output of genetic database queries that return a score for each reference sequence.

Another method of the invention for processing a biological sequence obtained from an assay comprises: converting base calls located in a predetermined list of positions within the biological sequence to N; and determining the ratio of single nucleotide polymorphisms in the biological sequence relative to a reference sequence. Each entry in the predetermined list of positions represents the capability of a substance hybridizing to a microarray used to generate the biological sequence. The substance is not the nucleic acid of a target pathogen.

BRIEF DESCRIPTION OF THE DRAWINGS

A more complete appreciation of the invention will be readily obtained by reference to the following Description of the Example Embodiments and the accompanying drawings.

Fig. 1 is a schematic of an algorithm representing the relationship of three main tasks and logic of subtasks associated with tasks. Task I carries out filtering and subsequence selection, and then determines what database records prototype sequences are most similar to. Task II figures whether prototype sequence identifications support a common organism identification. Task III does final examination and decisions of the detected organism from the microarray data. ProSeq: prototype sequence; SubSeq: subsequences; HybSeq: hybridized sequence.

Fig. 2 is a detailed schematic of the filtering subtask of Task 1. For each ProSeq, primers region were masked as N (ambiguous) calls, then UniRate, was calculated from the HybSeq. For ProSeqs which passed the UniRate requirement, a revised sliding window algorithm attempted to grow a SubSeq that could be used as a query to BLAST. The identity (start location in the ProSeq and length) of a successfully grown SubSeq was placed in a file for batch querying via BLAST.

Fig. 3 is a detailed schematic of the subtask of Task 1 responsible for organism identification for an individual SubSeq. Each SubSeq sent to BLAST returned a list of possible matches contained in a Return array that was sorted through to find best bit score/expect value pair (MaxScore). If the MaxScore was greater than MIN (10^{-6}), all returns that had this best Score were sorted into a new array Rank1. The detailed decision process described in the method section then the organism of the SubSeq was identified.

Fig. 4. is a schematic of the subtask of Task 1, which determines the organism, determined for a ProSeq based on the results found for its SubSeq. All of the SubSeq of a particular ProSeq are compared between to determine the two best scoring SubSeq. When there was a single SubSeq or one scored much better than the other, the ProSeq inherited the properties of that SubSeq. Otherwise the common taxonomic class was determined as described in the patent text.

Fig. 5. Alignment of the influenza A NA1 ProSeq and A/Weiss/43, A/PuertoRico/8/34 strains. Raw and filtered hybridization chip results of A/puertoRico/8/34 also shown. * indicate perfectly matched sequences.

MODES FOR CARRYING OUT THE INVENTION

In the following description, for purposes of explanation and not limitation, specific details are set forth in order to provide a thorough understanding of the present invention. However, it will be apparent to one skilled in the art that the present invention may be practiced in other embodiments that depart from these specific details. In other instances, detailed descriptions of well-known methods and devices are omitted so as to not obscure the description of the present invention with unnecessary detail.

As used herein, the term "sequence" refers to a nucleic acid sequence, such as DNA or RNA, or a protein sequence. As used herein, "base" and "base call" can refer to either nucleotide bases or amino acids. As used herein the term "taxonomic" can refer to any level or class of identification of a pathogen, including but not limited to, genus, species, strain, and substrain. As used herein the term "reporting" can include transmitting a signal from one system to another and producing a human-readable report of any form. All disclosed methods may be computer-implemented on apparatus having the means for performing the methods.

Disclosed is a new software expert system, Computer-Implemented Biological Sequence Identifier system 2.0 (CIBSI 2.0), that can successfully use resolved base sequence information from custom designed Affymetrix resequencing microarrays to provide a simple list of organisms that are detected. This algorithm addresses a shortcoming of previous methods by incorporating new features to completely automate pathogen identification. The single program is capable of making correct decisions for all 26 pathogens contained on the RPM v1 microarray (19,20,23), whether detected alone or in combinations, with improved sensitivity. Although the program is currently applied to resequencing microarrays, the methodologies developed remain generally applicable as only the first portion of the algorithm handles issues specific to microarrays while the remainder deals with sequences that are suitable for use as a query by the BLAST algorithm. In developing the general identification algorithm, issues specific to resequencing microarrays that complicate their use were identified and resolved. Because the entire decision process for what is detected has been automated, it is straightforward to test whether the

rules used to make identifications are rigorous and applicable to any pathogen. With this efficient program, resequencing based assays can provide a competitive method to test simultaneously for many possible pathogens, providing output that can be interpreted by a non-expert.

Amplification, Hybridization, and Sequencing determination

The details of the RPM v1 microarray design and the experimental methods have been discussed in previous work (19,20,23). The experimental microarray data used in the present analysis were obtained using a variety of purified templates and clinical samples using random and multiplexed amplification schemes. GCOS software v1.3 (Affymetrix Inc., Santa Clara, CA) was used to align and scan hybridized microarrays to determine the intensity of each probe in every probe set. Base calls were made based on the intensity data of each probe set using GDAS v3.0.2.8 software (Affymetrix Inc., Santa Clara, CA) which used an implementation of the ABACUS algorithm (11). The sequences were represented in FASTA format for later analysis steps.

A resequencing microarray (RPM v.1) was previously designed for detection and sequence typing of 20 common respiratory and 6 CDC category A biothreat pathogens known to cause febrile respiratory illness based on ProSeqs without relying on predetermined hybridization patterns (19,20,23). Approximately 4000 RPM v.1 experiments, performed using different amplification schemes, single and multiple pathogen targets, purified nucleic acids and clinical samples, were examined in order to develop the pathogen identification algorithm. Results using this algorithm with clinical samples, identified pathogens and purified nucleic acids are discussed in detail in other work (19,20,23). In all cases, the algorithm correctly identified the organism at a species or strain level, depending on the length of the ProSeqs represented on the RPM v.1. Some specific examples will be discussed to illustrate how the algorithm performs under a variety of conditions.

The CIBSI 2.0 program handled a hierarchy of three tasks (Fig. 1): (I) determine to which database records the detected organisms are most similar to, (II) determinate whether the identifications from separate targets support a common organism identification, and (III) determinate whether detected organisms belong to the target set that the assay was designed to detect or are related to close genetic near neighbors. Target pathogens are the organisms the assay was specifically designed to detect. As used herein, the sets of probes that represent reference sequence selected from target pathogen genomes are referred to as a Prototype Sequence or "ProSeq" for brevity. The set of resolved bases that result from hybridization of genomic material to a ProSeq is referred to as the hybridized sequence or "HybSeq". The HybSeq is split into possible subsequences or "SubSeqs". One portion of the algorithm deals with ProSeq-based organism identification and is handled in three steps: initial filtering of individual HybSeqs into SubSeqs suitable for sequence similarity comparisons, database querying of individual SubSeqs, and taxonomic comparison of BLAST returns for each SubSeq. At the next level, ProSeqs were compared to determine if they supported the same identified organism. In the final step, detected organisms were compared to the list of target pathogens the assay was designed for in order to determine if any were positively detected. The level of discrimination that a particular sample supported was automatically determined.

Filtering

An initial filtering algorithm, REsequencing Pathogen Identifier (REPI), was developed previously (20) and the general concepts with revisions were incorporated into the current (automated detection) algorithm used in the CIBSI 2.0 program. Filtering and subsequence selection were used to remove potential biasing caused by reference sequence choice, and by other sources (primers) as well as separate a HybSeq into fragments of

significance for faster searching. This was the first subtask of Task I in Fig. 1 and is shown schematically in detail in Figure 2. When PCR amplification was used, microarrays were hybridized in the presence of only primers to determine locations where they resulted in hybridization. Any portions of the ProSeqs that hybridized with the primers were masked as N calls so that the HybSeq did not contain biased information. For each ProSeq, the ratio of SNPs to the total number of unique base calls, UniRate, was calculated from the HybSeq. To eliminate HybSeqs with insufficient hybridization if the UniRate was ≥20 % (the SNP threshold), the ProSeq was considered negative for target organism detection. A UniRate of 20% indicated that on average there were 5 SNP per 25 bp. It is unrealistic to expect significant specific hybridization of 25 bp probes with this frequency of differences between an organism similar to the target pathogen and the reference sequence that the ProSeq is based on. This would end the filtering subtask and return to the Task I loop, which would examine the next ProSeq. For ProSeqs with a ratio < 20 %, a more detailed examination was performed. At each location of the HybSeq, a revised sliding window algorithm (20) attempted to grow a SubSeq that could be used as a query to BLAST. Initially, the first 20 bases (initial length) after a location were examined. If less than 60% of those bases were ambiguous, N, the SubSeq entered the extension stage. The SubSeq was extended one base at a time until the total content of unique base calls fell below 40% (unique-base threshold) or if a sliding window containing the last 21 bases had fewer than 4 unique base calls. This was different from the REPI algorithm in which only the sliding window of 20 bases was used and growth of a SubSeq stopped when less than 40 % of the windows content was unique base calls. At this point, the SubSeq was examined and trailing N calls were removed. At least one location having 7 contiguous unique base calls was required to meet the word size parameter of BLAST and to keep the SubSeq for further analysis. SubSeqs longer than 100 bases were accepted. For acceptance, SubSeqs ≤30 bases required at least 95 % unique base calls (not "N"). For SubSeqs with 30 to 100 bases, subsequence acceptance required at least VARI (("SubSeq length"-30)*0.2857 + 70) % unique bases. For SubSeqs ≥80 bases, The BLAST word size parameter was changed to 11 if it contained at least 11 contiguous bases. The identity (start location in the ProSeq and length) of a successfully grown SubSeq was placed in an entry in the SubSeq array that retained information associated with each SubSeq. This identity and the SubSeq were placed in a file for batch querying via BLAST. This procedure was repeated, by continuing from the end of the previous successful SubSeq, or in the case of failure, by continuing from the point the window was initially grown from until the end of the HybSeq. Upon completion, the algorithm returned to the Task I loop and performed the BLAST subtask.

Database query

The BLAST subtask performed a batch similarity search of a database using SubSeq as the queries. The BLAST program used was the NCBI Blastall –p blastn version 2.12 with a defined set of parameters. The masking of low complex regions was performed for the seeding phase to speed up the query; however, low complexity repeats were included in the actual scoring. The entire nucleotide database from NCBI acquired on Feb. 7, 2006 was used as the reference database. (Note that earlier images of the database were used during development but all experiments were rerun with the algorithm as described with the image of the database obtained on this date.) The default gap penalty and nucleotide match score were used. The nucleotide mismatch penalty, –q, parameter was set to -1 rather than the default. The results of any BLAST query with an expect value < 0.0001 were returned in tabular format from the blastall program. The information about each return (bit score, expect value, mismatches, length of match) was placed in the Return{hash key}{info} hash using the SubSeq identity as the hash key for further analysis.

Taxonomy-base Pathogen Identification for ProSeqs from SubSeqs

The next subtask of Task I carried out was the determination of SubSeq() states and is shown in Fig. 3. In order to present simple data and to ease the decision process, the information about every SubSeq was summarized by two parameters. The "identified organism" represented the taxonomic class of the organism and the "organism uniqueness" indicated the quality of the organism identification. The elements in the Return hash were examined and ranked by score array for each separate SubSeq() of a ProSeq. The Score array contained a pair of parameters, the bit score and the expect value, that had a fixed relationship for a given database. It was sometimes appropriate to use a ranking score that either accounted for the size of the database (expect value) or not (bit score). Elements in the Return hash could have the same scores so all the elements with the highest bit score/ lowest expect value (MaxScore) were retained in a separate array Rank1. The full taxonomic classifications of every element in Rank1 were retrieved from the NCBI taxonomy database also obtained on Feb. 7, 2006 (See earlier note). If MaxScore expect values were greater than MAX (currently 10^{-6}), the SubSeq() had its identified organism and organism uniqueness information both updated to null. When MaxScore was sufficiently small, the returns placed in Rank1 were examined. When Rank1 contained a single element, the SubSeq was assigned an organism uniqueness of SeqUniqu. When Rank1 contained multiple elements, the SubSeq was assigned an organism uniqueness of TaxUnique if all returns belonged to the same taxonomy class; otherwise, the SubSeq's organism state was set to TaxAmbig. The tasks outlined in Fig. 3 were applied to each SubSeq() of the ProSeq. In all cases, an identified organism was assigned to each SubSeq(), which represented the taxonomic class that was the common parent of all the elements within Rank1.

After each SubSeq was examined, the algorithm moved to the next task, which was to determine the identified organism of the ProSeq from the SubSeq (Fig. 4). If all the elements of SubSeq had an identified organism value of Null, then the ProSeq was negative and the next ProSeq was examined. When there was only a single element in SubSeq for a ProSeq or all the elements in SubSeq had the same identified organism, then a Result1 entry was created for the identified organism and its organism uniqueness was either TaxUnique if there were multiple SubSeq entries or it inherited the state of the single SubSeq entry. When there are many entries in SubSeq that had different identified organisms further analysis was performed. The SubSeqs were then re-sorted by MaxScore (bit score) so that elements with the highest two best Scores were SubSeq(1) and SubSeq(2). If SubSeq(1) had a Score that was $\geq 30\%$ (score ratio threshold) than SubSeq(2)'s Score, then the ProSeq inherited SubSeq(1)'s organism uniqueness and identified organism. Otherwise the ProSeq's organism state was TaxAmbig and the identified organism was the common parent taxonomic class of all the subsequences. If all subsequences were contained in only two taxonomic classes that were direct child and parent, the identified organism was the organism of the subsequences in the child class. The subtask covered in Fig. 4 was completed and the Task I loop continued. A list of ProSeq that had detected organisms was built up in the Result1 array.

Overall pathogen identification and positive calls

After Task I was completed, Task II (see Fig. 1) was used to examine the identified organism values listed in Result1 and grouped them together if they identified the same taxonomic class. Each entry in Result1 was examined and a new entry was created in Result2 if the identified organism did not appear in this list. In most cases, the entries of Result2 represented the individual organisms that were detected, but might still contain redundant information. Entries in Result2 with identified organisms where one was the taxonomic parent of the other might

actually represent the same pathogen. The same identification might not have occurred because the genomic target did not hybridize well to both ProSeq for a variety of possible reasons. Alternatively, two different but closely related organisms might both hybridize to the microarray.

Although it was difficult to relate results from separate ProSeqs to each other, Task III handled the final examination and decisions as currently implemented. The previous tasks were specifically implemented so that information about what a ProSeq was intended to detect was not considered. This allowed these lower cases to be capable of recognizing not just positive and negative but also cases that were indeterminate. In the final task, the algorithm considered whether the ProSeqs identified the organisms that they were designed to detect. Clear negative ProSeqs and ProSeqs that were indeterminate were considered negative for a target pathogen. The grouping of the ProSeqs for this was based upon the grouping already performed in task II. The entries of Result2 were looped over. The ProSeq of an entry was used to look up the pathogen in a table that it was targeted for. If the identified organism of the Result2 entry was the same as or a child of target pathogen taxonomic class, then the Pathogen() array was updated with a positive entry for that targeted pathogen. If the Pathogen() array was null for that pathogen, the pathogen level identified organism was that of the Result2() entry. If an entry was already placed in the pathogen, then further comparison was required. The Result2() and Pathogen entries were compared. If they had a direct child parent relationship, the identified organism of the Pathogen was the taxonomic class of the child. Otherwise the common parent taxonomic class was reported as the positively identified organism. In most cases where all the ProSeqs for a pathogen hybridized well, a fine level discrimination was reported. But if one or more ProSeqs hybridized less well, the reported positive target pathogen was only identified at the genus or species level. The results of all 3 tasks were reported so that manual re-examination was possible. Note that organisms identified in Task II that did not belong to target pathogens were reported as a non-target positive return. The details of what was identified in these cases required examination of Task II level results.

Pathogen Identification

Chlamydia pneumoniae samples with 10 to 1000 genome copes (via ref. 21 methods) were chosen to illustrate how pathogen detection and identification was done when multiple ProSeq were targeted for the same pathogen (21). RPM v.1 has three highly conserved ProSeqs selected from the genes encoding for the major outer membrane proteins VD2 and VD4, and the DNA-directed RNA polymerase (rpoB) gene. The HybSeqs from the different samples differed only in how many unique base calls as shown in Table 1. The percentage of the ProSeq called varied from 80 to 100% except for one case at a concentration of 10 that had only 11% of the rpoB ProSeq producing unique calls indicating that the detection limits of the assay have been reached is above this concentration. Table 1 listed the determinations made for the SubSeq and at the end of each Task for the various samples. The ProSeq from the different cases produced the same number of SubSeq. These SubSeq from different samples reported different bit scores for the same top ranked returns from BLAST. In fact VD2 and VD4 produced exactly the same results. The NCBI taxonomy database classified the returns into four distinct groups, which represented the *C. pneumoniae* taxonomic group and three child strain groups. AE001652, AE002167, AE017159, and BA000008 appeared in the returns of all the ProSeq for each sample, since they represented database entries of completely sequenced genomes. One rpoB SubSeq produced for its organism uniqueness, SeqUniqu. All other SubSeq were TaxAmbig as multiple returns from different taxonomic classes were returned. Since the VD2 and VD4 ProSeq each have a single SubSeq, Task I assigned the ProSeq the state of the SubSeq. For the rpoB ProSeq, the bit score of one SubSeq was large enough that the algorithm assigned that SubSeq's identification to the ProSeq.

Task II of the algorithm grouped all three ProSeqs together since they all had the same identified organism and TaxAmbig was assigned. The result of task III was positive for target pathogen *C. pneumoniae* and this decision was straightforward as all the ProSeqs agreed with each other and belonged to the same target pathogen taxonomic class. Although the rpoB ProSeq was SeqUniqu, this was not the final conclusion for Task II as the ProSeq that was SeqUniqu was not the child taxonomic group and other ProSeq were TaxAmbig. The three recognized substrains scored the same, which indicated that the sequence selected for the ProSeqs was very conserved and would not allow discrimination between the strains.

Table 1 – Algorithm decisions for *C. pneumoniae* at several concentrations for SubSeq, Task I, II, and III.

Genome Copies	ProSeq	Unique calls	# SubSeq	SubSeq organism identification and Uniqueness, Bit score	Task I	Task II	Task III
1000	VD2	89%	1	(G1)*C. pne*, **TA**, 145	*C. pne*, **TA**	*C. pne* **TA**	POSITIVE *C. pne*
	VD4	91%	1	(G1)*C. pne*, **TA**, 145	*C. pne* **TA**		
	rpoB	80%	2	(G2)*C. pne*, **SU**, 307 (G3)*C. pne*, **TA**, 73	*C. pne* **SU**		
100	VD2	100%	1	(G1)*C. pne*, **TA**, 164	*C. pne* **TA**	*C. pne* **TA**	POSITIVE *C. pne*
	VD4	97%	1	(G1)*C. pne*, **TA**, 156	*C. pne* **TA**		
	rpoB	80%	2	(G2)*C. pne*, **SU**, 343 (G3)*C. pne*, **TA**, 87	*C. pne* **SU**		
100	VD2	83%	1	(G1)*C. pne*, **TA**, 136	*C. pne* **TA**	*C. pne* **TA**	POSITIVE *C. pne*
	VD4	91%	1	(G1)*C. pne*, **TA**, 145	*C. pne* **TA**		
	rpoB	84%	2	(G2)*C. pne*, **SU**, 318 (G3)*C. pne*, **TA**, 82	*C. pne* **SU**		
10	VD2	100 %	1	(G1)*C. pne*, **TA**, 164	*C. pne* **TA**	*C. pne* **TA**	POSITIVE *C. pne*
	VD4	97%	1	(G1)*C. pne*, **TA**, 156	*C. pne* **TA**		
	rpoB	90%	2	(G2)*C. pne*, **SU**, 340 (G3)*C. pne*, **TA**, 89	*C. pne* **SU**		
10	VD2	100%	1	(G1)*C. pne*, **TA**, 164	*C. pne* **TA**	*C. pne* **TA**	POSITIVE *C. pne*
	VD4	93%	1	(G1)*C. pne*, **TA**, 148	*C. pne* **TA**		
	rpoB	11%	0	Null,Null	Null, Null		

(G1) J138 (BA000008), AR39 (AE002167),Tw-183 (AE017159),Cpne (M69230,AF131889,AY555078,M64064,AF131229,AF131230)
(G2) Cpne (S83995)
(G3) J138 (BA000008),AR39 (AE002167),Tw-183 (AE017159)
SU abbreviation for SeqUniqu
TA abbreviation for TaxAmbig

Influenza and Human Adenovirus (HAdV) were the only pathogens that had ProSeq selected that would permit detailed strain level discrimination as discussed in previous work (19,20,21). This previous work using manual analysis found that the microarray results were in excellent agreement with the conventional sequencing results for clinical samples. The results of running the CIBSI 2.0 program using the updated NCBI database on the raw microarray results were compared to the previous findings (Table 2). The identified organisms were not identical to the original findings due to the difference in database used. In fact, the conventional sequencing results that were submitted to NCBI from that work were found for every sample to be among the returns with the best Score. For 8 of 13 Influenza A and 3 of 12 Influenza B cases, the results of Task I and II found that the conventional sequencing was the single best return and hence the identified organism. Due to the large number of isolate sequences in the database for the *hemagglutinin* gene it was not surprising that in some cases a single unique entry

was not found. In each of the remaining 5 Influenza A samples, the other sequences returned differed by less than 0.2% from the conventional sequence. The fewer samples with unique isolate identifications for Influenza B were due to using an older reference sequence for the ProSeq, which allowed less hybridization to occur (19). This also meant that when multiple sequences were returned for a sample they represented greater genetic variation, up to 2%. This comparison represented only the algorithm analysis at the Task I level for the *hemagglutinin* (HA) ProSeq, which was the only region conventionally sequenced. There can be no comparison for Task III results, as the previous work did not attempt to obtain a consensus from multiple ProSeq. As a result of the current method of making Task III level identification, the organism reported at this level was less specific (H3N2 or Flu B) for every sample (Supplemental Table 1A and 1B). For HAdV samples, the algorithm also reproduced the finer scale discriminations that had been made previously by manual methods (not shown).

Table 2 – Pathogens identified from HA ProSeqs of Influenza A and B Clinical Sample.

Sample name	Strain identification by previous work (16)	GenBank accession no.	HA3 ProSeq identification by CIBSI 2.0	GenBank accession no.
A/Colorado/360/05	A/Nepal/1679/2004 (H3N2)	AY945284	A/Colorado/360/05*	DQ265717
A/Qatar/2039/05	A/Nepal/1727/2004 (H3N2)	AY945272	A/Qatar/2039/05	DQ265707
A/Guam/362/05	A/Nepal/1679/2004 (H3N2)	AY945264	A/Guam/362/05	DQ265715
A/Italy/384/05	A/Nepal/1727/2004 (H3N2)	AY945272	A/Italy/384/05	DQ265713
A/Turkey/2108/05	A/Nepal/1664/2004 (H3N2)	AY945265	A/Turkey/2108/05	DQ265718
A/Korea/298/05	A/Nepal/1727/2004 (H3N2)	AY945273	A/Korea/298/05	DQ265710
A/Japan/1337/05	A/Malaysia/2256/2004 (H3N2)	ISDN110616	A/Japan/1337/05*	DQ265712
A/Japan/1383/05	A/Malaysia/2256/2004 (H3N2)	ISDN110616	A/Japan/1383/05	DQ265711
A/Ecuador/1968/04	A/New York/17/2003 (H3N2)	CY001053	A/Ecuador/1968/04*	DQ265716
A/Iraq/34/05	A/Christchurch/178/2004 (H3N2)	ISDN110530	A/Iraq/34/05	DQ265714
A/Peru/166/05	A/Macau/103/2004 (H3N2)	ISDN64772	A/Peru/166/05	DQ265708
A/New York/2782/04	A/New York/391/2005 (H3N2)	CY002056	A/NewYork/2782/04s*	DQ265709
A/England/400/05	A/New York/227/2003 (H1N1)	CY002536	A/England/2005*	DQ265706
B/Peru/1324/04	B/Milano/66/04	AJ842082	B/Peru/1324/04s*	DQ265728
B/Peru/1364/04	B/Milano/66/04	AJ842082	B/Peru/1364/04s*	DQ265726
B/Colorado/2597/04	B/Texas/3/2002	AY139049	B/Colorado/2597/04s*	DQ265724
B/Japan/1905/05	B/Texas/3/2002	AY139049	B/Japan/1905/05s*	DQ265727
B/Japan/1224/05	B/Texas/3/2002	AY139049	B/Japan/1224/05s*	DQ265719
B/Alaska/1777/05	B/Texas/3/2002	AY139049	B/Alaska/1777/05s	DQ265730
B/England/1716/05	B/Texas/3/2002	AY139049	B/England/1716/05s	DQ265723
B/England/2054 /05	B/Texas/3/2002	AY139049	B/England/2054 /05*	DQ265722
B/Hawaii/1990/04	B/Tehran/80/02	AJ784042	B/Hawaii/1990/04	DQ265721
B/Hawaii/1993/04	B/Tehran/80/02	AJ784042	B/Hawaii/1993/04s*	DQ265720
B/Arizona/148/04	B/Tehran/80/02	AJ784042	B/Arizona/148/04*	DQ265725
B/Arizona/146/04	B/Tehran/80/02	AJ784042	B/Arizona/146/04*	DQ265729

* Multiple returns tied with this return

s HybSeq was split into multiple SubSeq

The next example of detection for the *Mycoplasma pneumoniae* pathogen demonstrated a case where there was only a single ProSeq for the target pathogen, which meant that the identified organism for Task I of the algorithm was automatically the result of Task II and was the only ProSeq considered in Task III for this targeted pathogen. This ProSeq was also not optimal for fine discrimination because it was selected from a highly conserved region (345 bp) of the *cytadhesin* P1 gene. 40 microarrays were tested with the same purified nucleic acid stock and

in every case *M. pneumoniae* or its one recognized substrain taxonomic database entries tied for MaxScore. To better understand these returns, the database sequences were examined and subdivided into three groups of sequences, A, B, and C, based on how well they matched the reference sequence used to make the ProSeq. The placement of the database entries into the three groups was determined from a CLUSTAL alignment of the sequences of this gene. This alignment confirmed that the database entries differed significantly more from each other in regions not represented by the ProSeq and contained sufficient variability that would have allowed finer discrimination. Members of Group A exactly matched the ProSeq and could not be distinguished between on the microarray. Similarly, members of group B matched the ProSeq except at the 199th position where the base called was C rather than T. Group C sequences contained a few database entries that were more variable and might be distinguished from other entries within the ProSeq. For the 40 experimental tests of *M. pneumoniae*, as much as 95% of the ProSeq hybridized, yet only 65% of the results had an unambiguous base call at the 199th position. When it was unambiguous, it always matched group B sequences. In the cases where an N base call was made at the 199th location, both groups A and B sequences were returned with the same score. Regardless of this, the target pathogen positively identified was *M. pneumoniae* for every sample tested.

These examples showed how decisions were made independent of whether single or multiple ProSeqs were dedicated to a target pathogen. They also illustrated that the level of discrimination possible was strongly determined by the quality of the selected ProSeq. It is possible that for some pathogens fine level discrimination is not required and the currently tested selections on RPM v1 would provide satisfactory information. The CIBSI 2.0 algorithm demonstrated its capability to automatically report the maximum level of discrimination that could be supported by the HybSeq information.

Genetic Near Neighbors

To demonstrate how the algorithm handled closely related genetic species, a sample of a non-targeted pathogen was considered. For Variola major virus, one of the biothreat pathogens on the RPM v.1, the validation runs demonstrated that Variola major virus DNA templates were always positively identified when detected. The array has two ProSeqs from *hemagglutinin* (VMVHA, ~500 bp) and cytokine response modifier B (VMVcrmB, ~300 bp) genes for Variola major virus detection. Table 3 shows the results for each ProSeq of 18 runs where a close neighbor, Vaccinia virus, was spiked into nasal wash at various concentrations. The percentage of the ProSeq that hybridizes is sufficient that if hybridization patterns were only considered one might assume that this tile is identifying the presence of its target. This would indicate that reference sequence selected was not the best choice. However, when the algorithm was applied none of the samples are in fact identified as Variola major or minor virus. Vaccinia was always one of the Orthopoxvirus species listed with the highest scores for VMVcrmB ProSeq, but in only 7 cases was it uniquely as the probable species detected. In 3 samples with the lowest concentration and fraction of VMVcrmB hybridizing, this ProSeq identified Variola major virus as one among many Orthopoxvirus species that could be the cause of the hybridization. The lower limit of detection for the amplification method used was between this concentration and the one above it. The VMVHA ProSeq made identifications of the Orthopoxvirus species in only two experiments and Variola major virus was listed as one of the tied best scoring returns. In both cases, VMVcrmB ProSeq specifically identified Vaccinia virus as the best match. The percentage of the hybridized ProSeq correlated with concentration of the sample.

Table 3 – Organism identification from Vaccinia sample on Variola Major virus ProSeqs.

	ProSeq			
	VMVCRMB		VMVHA	
CFU	%	identifies	%	identifies
$5*10^7$	77.9	Vaccinia virus	29.4	Orthopoxvirus
$5*10^7$	79.8	Vaccinia virus	25.7	Orthopoxvirus
$1.6*10^7$	79.4	Vaccinia virus	14.8	-
$1.6*10^7$	77.5	Orthopoxvirus*	24.5	-
$1.6*10^7$	76.8	Vaccinia virus	21.6	-
$1.6*10^7$	74.5	Orthopoxvirus*	17.3	-
$5*10^6$	77.9	Vaccinia virus	25.7	-
$5*10^6$	78.3	Orthopoxvirus*	22.0	-
$5*10^6$	73.0	Vaccinia virus	13.0	-
$5*10^6$	73.4	Orthopoxvirus*	7.8	-
$1.6*10^6$	75.3	Orthopoxvirus*	8.6	-
$1.6*10^6$	49.8	Vaccinia virus	6.6	-
$1.6*10^6$	65.5	Orthopoxvirus*	10.0	-
$1.6*10^6$	62.9	Orthopoxvirus*	8.2	-
$5*10^5$	58.4	Orthopoxvirus*	9.0	-
$5*10^5$	56.2	Orthopoxvirus	8.0	-
$5*10^5$	49.0	Orthopoxvirus	9.3	-
$5*10^5$	44.6	Orthopoxvirus	7.8	-

* - only neighbor species within Orthopoxvirus and not Variola major or Variola minor viruses CFU – colony forming units

Filtering

This example demonstrated the importance of the filtering portion of the algorithm by considering the HybSeqs of the ProSeqs for the H1N1 *neuraminidase* (NA1) and *matrix* genes from human Influenza A/Puerto Rico/8/34 (H1N1) strain. Filtering was necessary because sending the HybSeq of a ProSeq to BLAST in a single query can bias the scores against strains that have insertions or deletions relative to the ProSeq, especially when using BLAST parameters that maximized the use of base calls. The sliding window test was the portion of the algorithm that controlled filtering. If filtering were turned off, the entire HybSeq would be used in a single subsequence for two influenza ProSeqs that showed significant hybridization. A/Weiss/43 (H1N1) strain was identified as the most likely strain from the HybSeq of the NA1 ProSeq while the HybSeq of the matrix ProSeq correctly identified A/Puerto Rico/8/34. To better understand the source of biasing, CLUSTAL alignment of the NA1 gene of the two strains and the reference sequence used to make the ProSeq are shown in Fig. 5. The two strains showed 95% identity (67 mismatches in 1362 aligned bases); however, there was a stretch of 45 bases inserted in both A/Weiss/43 (SEQ ID NO.2) and the NA1 ProSeq (SEQ ID NO.1) compared to A/Puerto Rico/8/34 (SEQ ID NO.3). With the default filtering on, the NA1 ProSeq was split into five SubSeqs as the algorithm encountered large stretches of no calls. In Task I, the algorithm determined that three of the SubSeqs, which were shorter, had an identified organism of H1N1 as several isolates including A/Puerto Rico/8/34 tied with the best score, while the two other SubSeqs had an identified organism of only the A/Puerto Rico/8/34 strain as being the closest match. The organism identified by the NA1 ProSeq was A/Puerto Rico/8/34 because one of the SubSeqs had a much higher score. This ProSeq supported the same strain identification made in the *matrix* ProSeq. The organism identified was A/Puerto Rico/8/34 because two ProSeqs detected only that organism. The correct target pathogen was detected with filtering while without filtering the target pathogen level of identification would have

PCT/US06/22622
been Influenza A (H1N1 subtype) because two organisms were detected, A/Puerto Rico/8/34 and A/Weiss/43. Splitting the HybSeq into SubSeqs to remove biasing may reduce the level of identification as occurred for the three of the five SubSeqs in this case. The previous example for Vaccinia was another instance where the wrong species (Camel Pox or Callithrix jacchus) identification would occur if the filtering was not used. Clinical samples in Table 2 showed that HybSeqs split into multiple SubSeqs were capable of very specific identification.

As a minor point, when multiplex strategies rather than generic were used for amplification, it was necessary to perform additional filtering to remove potential biasing from the specific primers as described in the methods. Fig. 5 has the raw (SEQ ID NO.4) and mask filtered (SEQ ID NO.5) results for the hybridization of A/Puerto Rico/8/34 to show an example of this interference. There is in addition to the problem with biasing for the reasons described before a sequence of 18 bases present in the raw result that after filtering have been made N since they are in a location that interacts with the primers. If these base calls were included in the subsequences constructed, the query for the ProSeq would favor an incorrect strain.

The algorithm successfully provided pathogen identification to the maximum level of detail possible (species or strain) depending on the quality of each ProSeq. This identification capability requires minimal input on the identity of the pathogens making non-expert use feasible. The crucial feature incorporated that allowed complete automation was the use of taxonomic databases, which classify organisms into ordered groups and provide relationships between organism entries, allowing removal of redundancies, comparison of different related prototype sequences, and simplification of data presentation. This allows databases, i.e. NCBI, that are redundant and subject to minimal curation but which constantly receive updated and new sequence information to be used with great success. Although this has been demonstrated using only the NCBI databases, other databases or custom made ones could have easily been used, which might improve performance. The algorithm is capable of providing accurate identifications at all analysis levels for pathogens that are less variable or are represented by highly conserved ProSeqs. For more variable or rapidly mutating pathogens, e.g. Influenza A virus, Tasks I and II still provided accurate detailed identifications, but Task III was unable to report fine scale discrimination. The comparison of the conventionally sequenced Influenza virus gene sequences illustrated that the algorithm is capable of automatically adjusting for updates in databases. The algorithm demonstrated its capability to properly distinguish hybridizations on a ProSeq caused by the specified pathogen from those caused by genetically close (near neighbor) strains and did not make incorrect identifications, eliminating one potential cause of false positives. Filtering the raw hybridization results served to reduce the computation time, accounted for potential primer interference, and more importantly reduced potential biasing. This simple integrated algorithm provided sufficient and accurate identification, so that immediate use of the RPM v.1 or similar resequencing arrays and assay is possible.

Besides demonstrating the success of the CIBSI 2.0 program, the work involved in developing the algorithm allowed insight into the importance of proper ProSeq selection. The RPM v.1 was the first resequencing array designed specifically for multiple pathogen detection using database similarity searching and served as a prototype for this application. It has been demonstrated that a single ProSeq with as few as 100 bp, when designed correctly, can be sufficient to unambiguously identify an organism. However, it is clearly indicated that several longer ProSeqs provide better confirmation and more detailed information of a pathogen. The emphasis of the design to this point has been on capabilities that are generally applicable to any pathogen. Improving on performance of Task III may require more information about an individual pathogen and may have to be developed for each specific pathogen or class of pathogens. This information also may be required for the algorithm to identify which differences between a sample and database entries represent significant mutations. The hierarchal design of

the data analysis can make it easy to incorporate analysis that build upon the analysis already performed. The use of properly designed resequencing microarrays and this automated detection algorithm can provide a way forward to developing assays that can test for multiple organisms simultaneously while providing fine strain level discrimination giving access to information about detailed strain recognition, antibiotic resistance markers and pathogenicity. This will allow the analysis of partial sequence information from multiple organisms for applications such as differential diagnostics for illnesses with multiple potential causes (i.e. febrile respiratory illness), tracking of emergent pathogens, distinction of biological threats from harmless near genetic neighbors in surveillance applications, and for tracking the impact of co-infections or super infections. The concept of categorizing and reporting different degrees of identification depending on the quality of samples and set of target sequences is not limited to resequencing microarrays but is more generally applicable to any platform that is capable of returning sequence level calls that can be used to query a reference DNA database. As the trend for assays that test for multiple pathogens increases, automated analysis tools, such as this one, become more crucial for rapid identification in simple formats useful to the non-expert on a day to day basis.

Source code

Below is a PERL source code listing of an embodiment of the disclosed methods. The "overclinical" program is the top-level program that runs the other program. "fstorepi" performs the filtering, subsequence preparation, and query file preparation. This program uses in input file "primerhyb.dat" containing the predetermined list of positions to be changed to N. "runblast" performs the BLAST query. "dbparse" performs the taxonomic analysis. This program uses an input file "chip1pathogengroups" containing a list of target pathogens for each ProSeq.

overclinical

```
#!/usr/bin/perl
use File::Copy;
my $win=20;
my $debug=0;
my $nq="-usequ y";
$unam="";
$i=0;
while($i<=$#ARGV) {
  if(index($ARGV[$i],"-")+1) {
   if(substr($ARGV[$i],1) eq "file") {
    $unam=$ARGV[$i+1]; $i++;
   } elsif(substr($ARGV[$i],1) eq "debug") {
    $debug=$ARGV[$i+1]; $i++;
   }
   $i++;
  } else {
   $i++;
  }
}
if(-s $unam) {
 if (($l=index($unam,"."))+1) {
  $inp=substr($unam,0,$l);
 } else {
  $inp=$unam;
 }
 $rmfst=0;
 if(!($unam eq join("",$inp,".fst"))) { copy($unam,join("",$inp,".fst"));
$rmfst=1; }
$time="/usr/bin/ltime";
$ftime=join("",$inp,".tim");
 unlink $ftime;
```

PCT/US06/22622

```
system "./fstorepi -inp $inp -debug $debug";
$inp2=join("",$inp,"b1");
system "echo Blasting $inp2 >> $ftime";
system "$time -a -o $ftime ./runblast $nq -inp $inp2 -W 11";
$inp2=join("",$inp,"b2");
system "echo Blasting $inp2 >> $ftime";
system "$time -a -o $ftime ./runblast $nq -inp $inp2 -W 7";
system "echo Identifying $inp >> $ftime";
system "$time -a -o $ftime ./dbparse -inp $inp -debug $debug -targ 0 >
$inp.err 2>&1 ";
 unlink (join("",$inp,".fst")) if($rmfst);
} else {
print "Filename not found: $unam\n";
}
exit;


fstorepi

#!/usr/bin/perl
my $win=20;
my $probe=21;
$debug=0;
$unam="";
$mci=0;
$i=0;
while($i<=$#ARGV) {
 if(index($ARGV[$i],"-")+1) {
  if(substr($ARGV[$i],1) eq "inp") {
   $unam=$ARGV[$i+1]; $i++;
  } elsif(substr($ARGV[$i],1) eq "debug") {
   $debug=$ARGV[$i+1]; $i++;
  } elsif(substr($ARGV[$i],1) eq "snp") {
   $mci=$ARGV[$i+1]; $i++;
  }
  $i++;
 } else {
  $i++;
 }
}
$ch=join("",$unam,".fst");
if(!(-s $ch)) { print "Filename not Found: $ch\n"; exit;}
%hmi=();
if(open(INP,"primerhyb.dat")) {
 while($line=<INP>) {
  @a=split(" ",$line);
  $hmi{$a[0]}{$a[3]}{str}=$a[1];
  $hmi{$a[0]}{$a[3]}{beg}=$a[2];
 }
}
close(INP);
$ch=join("",$unam,".fst");
system("/bin/ls $ch > flist");
open(INP,"flist");
@mast=<INP>;
close(INP);
unlink "flist";
chomp @mast;
$search=">";
$Input="chip1tile.fasta";
open(INP,$Input) || die;
@inplist=();
while($line=<INP>) { $line=~ tr/\cM/ /; push(@inplist,$line);}
close(INP);
chomp @inplist;
@inchip=filterparse(\@inplist);
for $Input (@mast) {
 print "parsing $Input\n" if($debug);
 $Output=join("",">",substr($Input,0,-4),"b1.blai");
 $Outpu2=join("",">",substr($Input,0,-4),"b2.blai");
 open(INP,$Input);
```

PCT/US2006/022622

```
@inplist=();
while($line=<INP>) { $line=~ tr/\cM/ /; push(@inplist,$line); }
close(INP);
chomp @inplist;
@in=filterparse(\@inplist);
frcdet($mci,substr($Input,0,-4),\%hmi,@inchip,@in);
($olist,$olis2)=filter($win,$probe,\%hmi,@in);
open(OUT,$Output);
print OUT $olist;
close(OUT);
open(OUT,$Outpu2);
print OUT $olis2;
close(OUT);
}
exit;
sub filterparse {
 my ($inplist) = @_;
 $j=0;
 my @name=();
 my @val=();
 while($j<$#inplist+1) {
#  if(index($inplist[$j],"AFFX-TagIQ-EX")+1) {$j++;}
#  if(index($inplist[$j],"ATTIM")+1) {$j++;}
#  if(index($inplist[$j],"ATNAC")+1) {$j++;}
  if(index($inplist[$j],$search)+1) {
    $i=0;
    push(@name,join("`",split(/ /,$inplist[$j])));
    $j++;
    $temp="";
    while(!(index($inplist[$j],$search)+1) && $j<$#inplist+1) {
     $inplist[$j]=~ tr/\cM/ /;
     for ($inplist[$j]) {
      s/^\s+//; s/\s+$//; s/\s+/ /g;
     }
     $temp2=join("",$temp,$inplist[$j]);
     $temp=$temp2;
     $j++;
    }
    $temp=~ tr/catgN/CATGn/;
    push(@val,$temp);
  } else {
    $j=$j+1;
  }
 }
 return(\@name,\@val);
}
sub frcdet {
 my ($mci,$inp,$hmi,$nam3,$val3,$name,$val) =@_;
 my $output=join("",">",$inp,".frc");
 if($mci) { open(OUT,join("",">",$inp,".Bfl")); }
 for($j=0;$j<=$#{$name};$j=$j+1) {
  $l=index($$name[$j],":");
  $current=substr($$name[$j],1,$l-1);
  if(!(index($current,"AFFX")+1)) {
   print OUT "$current\n" if($mci);
   for($i=0;$i<=$#{$nam3};$i=$i+1) {
    if((index($$nam3[$i],$current)+1)) {$ku=$i; $i=$#{$nam3}+1; }
   }
   @ma=split(//,$$val3[$ku]);
   @mc=split(//,$$val[$j]);
   $mkey=substr($$name[$j],1,index($$name[$j],":")-1);
   if( exists ${$hmi}{$mkey} ) {
    for $skey (keys %{${$hmi}{$mkey}} ) {
     $len=length(${$hmi}{$mkey}{$skey}{str});
     if($len>2) {
      $beg=${$hmi}{$mkey}{$skey}{beg}-1;
      $end=$beg+$len;
      if($end>$#mc) {$end=$#use;}
      for($jm=$beg;$jm<$end;$jm++) { $mc[$jm]="N"; }
     }
    }
```

PCT/US06/22622

```
}
for($i=0;$i<=$#ma;$i++) {
 if(!(index($mc[$i],$ma[$i])+1)) {
  $mc[$i] =~ tr/CATG/catg/;
  if($mc[$i] ne "n" && $mc[$i] ne "N") {
   print OUT $i+1," $mc[$i]\n" if($mci);
  }
 }
}
$line=join("",@mc);
$$val[$j]=$line;
}
}
close(OUT)  if($mci);
if($mci) {
 open(OUT,join("",">",$inp,".Bfst"));
 $j=0;
 while($j<$#{$name}+1) {
#   print "$j $$name[$j]\n";
  print OUT "$$name[$j]\n";
  print OUT "$$val[$j]\n";
  $j++;
 }
 close(OUT);
}
open(OUT,$Output);
open(OUT2,join("",">",$inp,".frc2"));
$tocj=0; $toci=0;
$dir=0;
$outline="";
$outlin2="$Input\n";
for($j=0;$j<=$#{$name};$j=$j+1) {
 $l=index($$name[$j],":");
 $current=substr($$name[$j],1,$l-1);
 $cou=0; $cou2=0; $bcou=0; $l=length($$val[$j]);
 $cou= $$val[$j]=~ tr/n/n/;
 $cou2=$$val[$j]=~ tr/N/N/;
 $bcou=$$val[$j]=~ tr/c/c/;
 $bcou=$bcou+($$val[$j]=~ tr/t/t/);
 $bcou=$bcou+($$val[$j]=~ tr/a/a/);
 $bcou=$bcou+($$val[$j]=~ tr/g/g/);
 $l=$l-cou2;
 $cou=$l-$cou;
 $fr=$cou/$l*100;

$temp=join("",$outline,sprintf("%s`%s`%f`%g`%g`%g\n",$inp,$current,$fr,$cou,$
bcou,$l));
  $outline=$temp;
  if($fr>20 && 100*$bcou/($fr*$l)<.25) {
   $temp=join("",$outlin2,sprintf(" %14s %5.1f %6g
%6g\n",$current,$fr,$bcou,$l));
   $outlin2=$temp;
  }
 }
 print OUT $outline;
 print OUT2 $outlin2;
 close(OUT);
 close(OUT2);
 unlink join("",$inp,".frc2");
 return;
}
sub filter {
 my ($win,$probe,$hmi,$name,$val) =@_;
 my $olist=""; my $olis2="";
 for($jm=0;$jm<=$#{$name};$jm++) {
  my $tmolist=""; my $tmolis2="";
  if(index($$name[$jm],"AFFX-TagIQ-EX")+1) {$$val[$jm]="";}
  if(index($$name[$jm],"ATTIM")+1) {$$val[$jm]="";}
  if(index($$name[$jm],"ATNAC")+1) {$$val[$jm]="";}
  my @nuse=();
  $$val[$jm]=~ tr/CTAGn/ctagN/;
```

PCT/US06/22622

```
my @use=split(//,$$val[$jm]);
my $nummat=0;
$mkey=substr($$name[$jm],1,index($$name[$jm],":")-1);
if( exists ${$hmi}{$mkey} ) {
 for $skey (keys %{$$hmi{$mkey}} ) {
  $len=length(${$hmi}{$mkey}{$skey}{str});
  if($len>2) {
   $beg=${$hmi}{$mkey}{$skey}{beg}-1;
   $end=$beg+$len;
   if($end>$#use) {$end=$#use;}
   for($j=$beg;$j<$end;$j++) { $use[$j]="N"; }
  }
 }
}
for($j=0;$j<=$#use;$j++) {
 if(!($use[$j] eq "N")) { push(@nuse,"1"); $nummat++;
 } else { push(@nuse,"0");}
}
my $pri=0;
if($nummat>20) {
 for($j=0;$j<=$#use;$j++) {
#   if($nuse[$j]==1 && $nuse[$j+1]==1) {
  if($nuse[$j]==1) {
   my $current=substr($$val[$jm],$j,$win);
   my $score=1;
   my $zscore;
   for($k=$j+1;$k<$j+$win;$k++) { $score=$score+$nuse[$k]; }
#    if($score>4 && !(index($current,"NNNN")+1)) {
   if($score>4) {
    my $len=$win;
    $k=$j+($win-1);
    if($probe>$win) {
     while($len<$probe&&$score/$len>.4) {
      $k++; $len++;
      $score=$score+$nuse[$k];
     }
     $zscore=$score;
    } else {
     $zscore=$score;
     for($jl=$k-$win;$jl<$k-$probe;$jl++) {$zscore=$zscore-$nuse[$jl];}
    }
    while($score/$len>.4&&$zscore>3 ) {
     $k++; $len++;
     $score=$score+$nuse[$k];
     $zscore=$zscore-$nuse[$k-$probe]+$nuse[$k];
    }
    $len--; $jl=$j+$len-1;
    if($nuse[$jl]==1 && ($nuse[$jl-1]+$nuse[$jl-2]+$nuse[$jl-3]+$nuse[$jl-
4])<1) {
     $jl--; $len--;
    }
    while($nuse[$jl]<1) {
     $jl--; $len--;
     if($nuse[$jl]==1 && ($nuse[$jl-1]+$nuse[$jl-2]+$nuse[$jl-3]+$nuse[$jl-
4])<1) {
      $jl--; $len--;
     }
    }
    $current=substr($$val[$jm],$j,$len);
    if( $current =~ m/[ctgaCTGA]{7}/ ) {
     if($len>100) {
      $pri++;
      if($score>80 && $current =~ m/[ctgaCTGA]{11}/) {
       $tmolist=join("",$tmolist,$$name[$jm],":",$j,"\n",$current,"\n");
      } else {
       $tmolis2=join("",$tmolis2,$$name[$jm],":",$j,"\n",$current,"\n");
      }
      $j=$k-1;
     } elsif($len>30) {
      if( $score/$len >= (.95-.25*($len-30)/(70.))) {
       $pri++;
```

PCT/US06/22622

```
            if($score>80 && $current =~ m/[ctgaCTGA]{11}/) {
             $tmolist=join("",$tmolist,$$name[$jm],":",$j,"\n",$current,"\n");
            } else {
             $tmolis2=join("",$tmolis2,$$name[$jm],":",$j,"\n",$current,"\n");
            }
            $j=$k-1;
           }
          } elsif($len>20) {
           if($score/$len>=.95) {
            $pri++;
            $tmolis2=join("",$tmolis2,$$name[$jm],":",$j,"\n",$current,"\n");
            $j=$k-1;
           }
          } else {
           $j=$k-($win-1);
          }
         }
        }
       }
      }
     }
     if($pri>1) {
      $olist=join("",$olist,$tmolist);
      $olis2=join("",$olis2,$tmolis2);
     } elsif($pri>0) {
      $olist=join("",$olist,$tmolist);
      my @a=split("\n",$tmolis2);
      if( $a[1] =~ m/[ctgaCTGA]{11}/) {
       $olist=join("",$olist,$tmolis2);
      }
     }
    }
    return ($olist,$olis2);
   }

   runblast

   #!/usr/bin/perl
   # -*- perl -*-
   #
   # perl script to handle filtering originally implemented in JAVA REPI and
   carry out blastall
   # Created 7/12/2005 - For Xserve cluster so execution of
   #                          blast search done through SGE job queue
   #
   use File::Copy;
   {
    $user=getlogin();
    my $debug=0;
    my $noque=1;
    my $srcdir="./";
    my $tmpdir="/common/scratch/repi/";
    my
   @in,$eltime,$Input,$infile,$outfile,$insge,$outsge,$errsge,$idsge,$cmde,$cmdf
   ,$cmdw,$cmda,$cmdo,$cmd,$cmdb,$cmdfi,$err;
    $ENV{SGE_ROOT}="/common/sge";
    # Extract the command line options for this script the ones to be passed to
   blastall.
    my %param=();
    $cmdblast="/common/bin/blastall";
    $param{e}="0.0001";
    $param{F}=join("","\\\"","m D","\\\" ");;
    $param{W}="7";
    $param{m}="8";
    $param{r}="1";
    $param{q}="-1";
    $param{b}="750";
    $param{d}="/common/data/nt";
    my $php=0;
```

PCT/US2006/022622

```
my $repiparse=0;
my $win=0;
$i=0;
while($i<=$#ARGV) {
 if(index($ARGV[$i],"-inp")+1) {
  $Input=$ARGV[$i+1]; $i++;
 } elsif (index($ARGV[$i],"-debug")+1) {
  $debug=$ARGV[$i+1]; $i++;
 } elsif (index($ARGV[$i],"-usequ")+1) {
  $noque=0 if(index($ARGV[$i+1],"y")+1 || index($ARGV[$i+1],"Y")+1); $i++;
 } elsif (index($ARGV[$i],"-")+1) {
  if(substr($ARGV[$i],1) eq "F") {
   if($ARGV[$i+2] eq "-U") {
    $param{substr($ARGV[$i],1)}=join("","\\\"",$ARGV[$i+1],"\\\"
",$ARGV[$i+2]); $i=$i+2;
   } else {
    $param{substr($ARGV[$i],1)}=join("","\\\"",$ARGV[$i+1],"\\\" "); $i++;
   }
  } else {
   $param{substr($ARGV[$i],1)}=$ARGV[$i+1]; $i++;
  }
 }
 $i++;
}
print "beginning repiv2 with input filename $Input\n" if($debug);
if($noque) {
 $infile=join("",$Input,".blai");
 $outfile=join("",$Input,".blao");
 $cmd="$cmdblast -p blastn -i $infile -o $outfile";
 for $mkey (keys %param) {
  $cmd=join("",$cmd," -",$mkey," ",$param{$mkey});
 }
# Run command
 system"$cmd";
} else {
# construct names of required files and remove preexisting versions of files
# files for final report
 my $errfile=join("",$srcdir,$Input,".err");
 unlink $errfile;
 unlink join("",$srcdir,$Input,".blao");
# files for blast command
 $infile = join("",$tmpdir,$Input,".itmp");
 $outfile = join("",$tmpdir,$Input,".otmp");
 unlink ($infile,$outfile);
# files for batch queue
 $insge=join("","cgisge-",$Input);
 opendir (DIR, $tmpdir);
 $a=join("","cgisge-",$Input);
 @files=grep { /$a/ } readdir(DIR);
 unlink ($tmpdir.$_) for (@files);
 closedir(DIR);
 $insge=join("",$tmpdir,$insge);
 $idsge=join(".",$insge,"id");
# check for error in parameters list
#  errordie($errfile,$err,$php) if ($err);
 if(!( -s join("",$srcdir,$Input,".blai"))) {
  copy(join("",$srcdir,$Input,".blai"),join("",$srcdir,$Input,".blao"));
  exit;
 }
# create input blast file
 copy(join("",$srcdir,$Input,".blai"),$infile);
# join list of variable parameters to Basic command with input-output file
names
 $cmd="$cmdblast -p blastn -i $infile -o $outfile";
 for $mkey (keys %param) {
  $cmd=join("",$cmd," -",$mkey," ",$param{$mkey});
 }
# Run command
# additional code required to generate results working through sge
 open (OUT, ">$insge") or errordie($errfile,"Failed to write batch queue
script\n  $insge\n",$php);
```

PCT/US06/22622

```
print OUT "#!/usr/bin/perl \n";
print OUT "system(\"$cmd\"); \n";
print OUT "open(OUT,\">$idsge\"); \n";
print OUT "print OUT \"\$ENV{JOB_ID}\"; \n";
print OUT "close OUT; \n";
close OUT;
($sec,$min,$hour,$day,$month,$year,$dweek,$dyear,$dsave)=localtime(time);
$eltime= join("","Summary of results\n refer to .blao file for raw blast
results\n Start ",$hour,":",$min,":",$sec);
$i=`/common/sge/bin/darwin/qsub -sync y -o $tmpdir -e $tmpdir -S
/usr/bin/perl $insge`;
@a=split(" ",$i); $id=$a[2];
$outsge=join("",$insge,".o",$id);
$errsge=join("",$insge,".e",$id);
unlink ($insge,$idsge);
($sec,$min,$hour,$day,$month,$year,$dweek,$dyear,$dsave)=localtime(time);
$eltime= join("",$eltime," END ",$hour,":",$min,":",$sec,"\n");
# print the output to the file
if ( -e $outfile ) {
 move($outfile,join("",$srcdir,$Input,".blao"));
 # remove input and output files
 unlink ($outsge,$errsge,$infile,$outfile);
} else {
 open(OUTFILE,"$errsge"); my @errput = <OUTFILE>; close OUTFILE;
 $err= join "", @errput;
 # remove input and output files
 errordie($errfile,$err,$php);
}
}
exit;
}
sub errordie {
 my ($file,$err,$php) = @_;
 open (INP,">$file") or die "can\'t open file error file";
 print INP "ERROR:\n $err";
 exit;
}


dbparse

#!/usr/bin/perl
# -*- perl -*-
#
use strict;
use warnings;
use DBI;
my $usetarsuc=1;
my $bound=1e-6;
my $debug=0;
my $unam="";
my $i=0;
while($i<=$#ARGV) {
 if(index($ARGV[$i],"-")+1) {
  if(substr($ARGV[$i],1) eq "inp") {
   $unam=$ARGV[$i+1]; $i++;
  } elsif(substr($ARGV[$i],1) eq "debug") {
   $debug=$ARGV[$i+1]; $i++;
  } elsif(substr($ARGV[$i],1) eq "targ") {
   $usetarsuc=$ARGV[$i+1]; $i++;
  }
  $i++;
 } else {
  $i++;
 }
}
 my $chippatho="chip1pathogengroups";
 open (INP,$chippatho) or die "Failed to open input file $chippatho\n";
 my %patho=(); my $line;
 while($line=<INP>) {
  $line=~ tr/\cM/ /;
```

```
chomp $line;
my @a=split("`",$line);
for(my $i=1;$i<=$#a;$i++) {
 my @b=split(",",$a[$i]);
 my $c =join("",",",$b[1],",");
 $patho{$a[0]}{$c}{name}=$b[0];
 $patho{$a[0]}{$c}{stat}=$b[2];
}
}
close(INP);
$chippatho="targets.runs";
open (INP,$chippatho) or die "Failed to open input file $chippatho\n";
my $temptar=0;
my %target=();
while($line=<INP>) {
 $line=~ tr/\cM/ /;
 chomp $line;
 my @a=split("`",$line);
 for(my $i=1;$i<=$#a;$i++) {
  my @b=split(",",$a[$i]);
  my $c =join("",",",$b[1],",");
  $target{$a[0]}{$c}{name}=$b[0];
  $temptar=1 if(index($unam,$a[0])+1);
 }
}
if($usetarsuc && !$temptar ) {
  $usetarsuc=0; print "have not indicated target for $unam\n";
} elsif(!$usetarsuc && $temptar ) {
  print "have indicated target for $unam but not using\n";
}
close(INP);
$chippatho="chip1taxexempt";
my %taxexempt=();
if(open (INP,$chippatho)) {
 while($line=<INP>) {
  $line=~ tr/\cM/ /;
  chomp $line;
  my @a=split("`",$line);
  $taxexempt{$a[0]}=$a[1];
 }
 close(INP);
} else {
 print "No $chippatho file\n";
}
my @mast=(join("",$unam,".blao"));
my $dbh=openconnection();
for my $Input (@mast) {
 my @a= split("/",substr($Input,0,-5));
 my $SInput=$a[$#a];
 if(! exists $target{$SInput}) {
  $target{$SInput}{1}{taxid}=-999;
  $target{$SInput}{1}{name}="unknown";
 }
 print "parsing $Input \n" if($debug);
 outfilter(substr($Input,0,-5),$SInput,my
$eltime,\%patho,\%target,\%taxexempt,$bound) ;
}
closeconnection($dbh);
exit;
sub outfilter {
 use strict;
 my ($Input,$SInput,$eltime,$patho,$target,$taxexempt,$bound) = @_;
 my %ha=();
 my %iha=();
 my %liac=();
 my $Output=join("",">",$Input,".rdb");
 my %frclist=(); my $line;
 if(open(INP,join("",$Input,".frc")) ) {
  while($line=<INP>) {
   $line=~ tr/\cM/ /;
   chomp $line;
```

PCT/US06/22622

```
        my @a=split(" ",$line);
        $frclist{$a[1]}{per}=$a[2];
        $frclist{$a[1]}{mat}=$a[3];
        $frclist{$a[1]}{mis}=$a[4];
        $frclist{$a[1]}{len}=$a[5];
       }
       close(INP);
      }
      unlink "flist";
      unlink "flisterr";
      my $ch=join("",$Input,"b?.blai ",$Input,".blai");
      system("(/bin/ls $ch > flist) >& flisterr");
      open(INP,"flist");
      my @mast=<INP>;
      close(INP);
      unlink "flist";
      unlink "flisterr";
      chomp @mast;
      my @inplist=();
      if(!@mast) { print "Files for $Input not found\n"; exit;}
      for my $Inp  (@mast) {
       open(INP,$Inp);
       while($line=<INP>) {
        $line=~ tr/\cM/ /;
        push(@inplist,$line);
       }
       close(INP);
      }
      chomp @inplist;
      my $j=0;
      my $sear=">";
      while($j<=$#inplist) {
       if(index($inplist[$j],">")+1) {
         my $mkey=substr($inplist[$j],1,index($inplist[$j],":")-1);
         my @a=split(" ",$inplist[$j]);
         my @b=split(":",$a[0]);
         my $mnkey=0;
         $mnkey=$b[2] if($#b>1);
         $j++;
         my $temp="";
         my $temp2=$temp;
         while($j<$#inplist+1 && !(index($inplist[$j],$sear)+1) ) {
          $inplist[$j]=~ tr/\cM/ /;
          for ($inplist[$j]) {
           s/^\s+//; s/\s+$//; s/\s+/ /g;
          }
          $temp2=join("",$temp,$inplist[$j]);
          $j++;
          $temp=$temp2;
         }
         $temp=~ tr/catgN/CATGn/;
         $iha{$mkey}{$mnkey}= $temp;
       } else {
         $j++;
       }
      }
      $ch=join("",$Input,"b?.blao ",$Input,".blao");
      system("(/bin/ls $ch > flist) >& flisterr");
      open(INP,"flist");
      @mast=<INP>;
      close(INP);
      unlink "flist";
      unlink "flisterr";
      chomp @mast;
      @inplist=();
      for my $Inp  (@mast) {
       open(INP,$Inp);
       while($line=<INP>) {
        $line=~ tr/\cM/ /;
        push(@inplist,$line);
       }
```

PCT/US06/22622

```
close(INP);
}
chomp @inplist;
$j=0;
while($j<=$#inplist) {
  my $mkey=substr($inplist[$j],0,index($inplist[$j],":"));
  my @a=split(" ",$inplist[$j]);
  my @b=split(":",$a[0]);
  my $mnkey=0;
  $mnkey=$b[2] if($#b>1);
  my @temp=split(/\|/,$a[1]);
  my $skey=$temp[$#temp];
  $line= $skey;
  my $gi=-1;
  if(index($a[1],"gi\|")+1) {
   $gi=$temp[1];
   $line=$temp[3];
   $skey=$gi;
   if((my $i=index($skey,"\."))+1) { $skey=substr($skey,0,$i); }
  }
  chomp $skey;
  $ha{$mkey}{$mnkey}{"$skey"}{line}=$line;
  $ha{$mkey}{$mnkey}{"$skey"}{scor}=$a[11];
  $ha{$mkey}{$mnkey}{"$skey"}{eval}=$a[10];
  $ha{$mkey}{$mnkey}{"$skey"}{gi}=$gi;
  my $temp=substr($iha{$mkey}{$mnkey},($a[6]-1),($a[7]-$a[6]+1));
  $a[6]=$a[6]+$mnkey;
  $a[7]=$a[7]+$mnkey;
  shift @a;
  $a[0]=$a[1]; $a[1]=$a[2]; $a[2]=$a[3];
  $a[3]= $temp=~ tr/nN/nN/;
  $a[3]=$a[2]-$a[3];
  $ha{$mkey}{$mnkey}{"$skey"}{out}=join(" "," ",@a);
  $j++;
}
foreach my $mkey (keys %ha) {
 my ($sec,$min,$hour,$day,$month,$year,$dweek,$dyear,$dsave);
 my @low=();
 foreach my $mnkey (sort keys %{$ha{$mkey}}) {
  foreach $line (sort keys %{$ha{$mkey}{$mnkey}}) {
   if($ha{$mkey}{$mnkey}{$line}{gi}<0) {
    $liac{$line}="-1`No database entry";
   } else {
    push(@low,$ha{$mkey}{$mnkey}{$line}{gi}) if(! exists $liac{$line});
   }
  }
 }
 if(1) {
  for my $giid (@low) {
   my $taxid= taxfromgi($giid);
   if( $taxid >-1 ) {
    my $ch = taxparents($taxid);
    if(!(index($ch,",28384,")+1)) {
     my @a=split(/,/,substr($ch,1,-1));
     my $comname= `/common/bin/fastacmd -d /common/data/nt -s $giid  -L 0,3`;
     @a= split(" ",$comname); shift @a; $comname=join(" ",@a);
     @a=split(";",$comname);
     my @b=split(",",$a[0]);
     $comname= $b[0];
     $liac{$giid}=join("`",$taxid,taxname($taxid),$comname,$ch);
    }  else {
     $liac{$giid}=-2;
    }
   } else {
#     print "no tax for gi $giid\n";
    $liac{$giid}="-1`No database entry";
   }
  }
 } else {
  use Bio::Seq;
  use Bio::DB::GenBank;
```

PCT/US06/22622

```
my $gb = new Bio::DB::GenBank();
my $seqio=();
$seqio = $gb->get_Stream_by_id([@low]);
while((my $seq1=$seqio->next_seq())) {
 if($seq1->alphabet() ne 'protein') {
  $line=$seq1->primary_id();
  my $spe=$seq1->species();
  my $ch=join(";",$spe->classification());
  if(!(index($ch,"artificial")+1)) {
   my @a=split(";",$seq1->desc());
   my @b=split(",",$a[0]);
   my $comname= $b[0];
   my $taxid=$spe->ncbi_taxid();
   $liac{$line}=join("`",$taxid,$spe->common_name(),$comname);
  } else {
   $liac{$line}=-2;
  }
 }
}
}
for $line (@low) {
 if(!(exists $liac{$line})){
  $liac{$line}="-1`No database entry";
 }
}
}
open(OUT,$output);
my ($sec,$min,$hour,$day,$month,$year,$dweek,$dyear,$dsave)=localtime(time);
$eltime= join("","Time ",$hour,":",$min,":",$sec," Date
",$month+1,"/",$day,"/",$year+1900,"\n");
my $ltime=
sprintf("%02g:%02g:%02g;%02g/%02g/%4g",($hour,$min,$sec,$month+1,$day,$year+1
900));
print OUT $eltime;
print OUT " (Probe name`Experiment name`run version`taxid`Common
name`Description`pathogen match`target match`Accension number`gi
number`%identity`alignment length`mismatches`nonN
nismatches`gaps`st.start`st.end`fs.start`fs.end`e-value`,bit
score`rank`time;Date)\n";
my %subtile=(); my %tilerslt=();
foreach my $mkey (keys %ha) {
 foreach my $mnkey (sort keys %{$ha{$mkey}}) {
  my %alkey=();
  my $currank=2000000000000; my $rank=0;
  my $maxrank=0; my $scale=1.0;
  foreach $line (sort {   $ha{$mkey}{$mnkey}{$b}{scor} <=>
$ha{$mkey}{$mnkey}{$a}{scor}} keys %{$ha{$mkey}{$mnkey}}) {
   $maxrank=$ha{$mkey}{$mnkey}{$line}{scor}; last;
  }
  foreach $line (sort {   $ha{$mkey}{$mnkey}{$b}{scor} <=>
$ha{$mkey}{$mnkey}{$a}{scor}} keys %{$ha{$mkey}{$mnkey}}) {
   my @a=split("  ",$ha{$mkey}{$mnkey}{$line}{out});
   my $out1="$a[1]`$a[2]`$a[3]`$a[5]`$a[6]`$a[7]`$a[8]`$a[$#a-1]`$a[$#a]";
   my $out2=join("`",@a);
   my @taxinfo=split("`",$liac{$line});
   my $tilsuc=-1;
   my $tarsuc =0;
   if(exists $liac{$line} && $taxinfo[0] > -1) {
    my $taxin = $taxinfo[3];
    for my $tarkey (sort
{${$patho}{$mkey}{$a}{stat}<=>${$patho}{$mkey}{$b}{stat}}keys
%{${$patho}{$mkey}}) {
     if(index($taxinfo[3],$tarkey)+1) {
      $tilsuc=${$patho}{$mkey}{$tarkey}{stat};
#      $taxin =${$patho}{$mkey}{$tarkey}{name};
     last;
     }
    }
    if($usetarsuc) {
     for my $tarkey (keys %{${$target}{$SInput}}) {
      if(index($taxinfo[3],$tarkey)+1) {
```

```
          $tarsuc =1 if(substr($tarkey,1,-1)>0);
          last;
         }
        }
       }
       if($ha{$mkey}{$mnkey}{$line}{scor}<$currank) {
        $rank++;
        if($currank*$scale<$ha{$mkey}{$mnkey}{$line}{scor}) {
         $currank=$currank*$scale;
        } else {
         $currank=$ha{$mkey}{$mnkey}{$line}{scor};
        }
       }
       if($rank ==1) {$rank++ if($ha{$mkey}{$mnkey}{$line}{eval}>$bound);}
       my $uni=0;
       if(exists $alkey{$rank}) {
        if(exists $alkey{$rank}{$taxin}) {
         $alkey{$rank}{$taxin}{count}++;
$alkey{$rank}{$taxin}{out}=join(",",$alkey{$rank}{$taxin}{out},$ha{$mkey}{$mn
key}{$line}{line});
        } else {
         $uni=1;
         $alkey{$rank}{$taxin}{count}++;
$alkey{$rank}{$taxin}{out}="$taxinfo[1]`$taxinfo[2]`$taxinfo[3]`$tilsuc`$out1
`$ha{$mkey}{$mnkey}{$line}{line}";
        }
       } else {
        $uni=1;
        $alkey{$rank}{$taxin}{count}++;
$alkey{$rank}{$taxin}{out}="$taxinfo[1]`$taxinfo[2]`$taxinfo[3]`$tilsuc`$out1
`$ha{$mkey}{$mnkey}{$line}{line}";
       }
       print OUT
"$mkey`$SInput`primary`$taxinfo[0]`$taxinfo[1]`$taxinfo[2]`$tilsuc`$tarsuc`$h
a{$mkey}{$mnkey}{$line}{line}`$ha{$mkey}{$mnkey}{$line}{gi}`$out2`$rank`$uni`
$ltime\n";
      }
     }
# subsequence determinations
     $rank=1;
     if( exists $alkey{$rank} ) {
      my $detect="-1";
      my @clip=(); my $tcomp;
      for my $key2 (keys %{$alkey{$rank}} ) {
       my @a=split(/`/,$alkey{$rank}{$key2}{out});
       my @b=split(/,/,$a[2]);
       if( !exists ${$taxexempt}{$b[$#b]} ) {
        if(@clip) {
         my $i=0;
         if($#b <= $#clip) {
          if(($detect ne "TAXAMBIG") && ($b[$#b] == $clip[$#b])) {
           $detect="TAXUNIQU";
          } else {
           while(($i<=$#b && $i<=$#clip) && $b[$i] eq $clip[$i]) { $i++; }
           splice(@clip,$i);
           $detect="TAXAMBIG";
          }
         } else{
          if ( $detect ne "TAXAMBIG" && ($b[$#clip] == $clip[$#clip])) {
           $detect="TAXUNIQU";
           @clip=split(/,/,$a[2]);
          } else {
           while(($i<=$#b && $i<=$#clip) && $b[$i] eq $clip[$i]) { $i++; }
           splice(@clip,$i);
           $detect="TAXAMBIG";
          }
         }
        } else {
```

PCT/US06/22622

```
            @clip=split(/,/,$a[2]);
            $detect="SEQUNIQU";
            $detect="TAXUNIQU" if($alkey{$rank}{$key2}{count} > 1);
          }
        }
      }
      $subtile{$mkey}{$mnkey}{detect}=$detect;
      $subtile{$mkey}{$mnkey}{tax}=join("",join(",",@clip),",");
      $subtile{$mkey}{$mnkey}{out}=join("","    Subseq. of $mkey $detect
",taxname($clip[$#clip])," St:$mnkey\n");
      $subtile{$mkey}{$mnkey}{out}=join("",$subtile{$mkey}{$mnkey}{out},"
Rank $rank ");
      my $tempout="";
      my $tempou2="";
      for my $key2 (keys %{$alkey{$rank}} ) {
       my @a=split(/`/,$alkey{$rank}{$key2}{out});
       $subtile{$mkey}{$mnkey}{name}=$a[1] if(!exists
$subtile{$mkey}{$mnkey}{name});
       $subtile{$mkey}{$mnkey}{bit}=$a[12] if(!exists
$subtile{$mkey}{$mnkey}{bit});
       $tempou2="Bases:$a[4] MIS:$a[5] SNP:$a[6] Range:$a[7]-$a[8] E:$a[11]
Bit:$a[12]\n";
       my $name = $a[0]; my $out="Match:$a[3] Range:$a[9]-$a[10]\n
$a[13]";
       $tempout=join("",$tempout,"      $name Num:$alkey{$rank}{$key2}{count}
$out\n");
      }

$subtile{$mkey}{$mnkey}{out}=join("",$subtile{$mkey}{$mnkey}{out},$tempou2,$t
empout);
      $rank=2;
      $subtile{$mkey}{$mnkey}{out}=join("",$subtile{$mkey}{$mnkey}{out},"
Rank $rank ");
      $tempout="";
      for my $key2 (keys %{$alkey{$rank}} ) {
       my @a=split(/`/,$alkey{$rank}{$key2}{out});
       $tempou2="Bases:$a[4] MIS:$a[5] SNP:$a[6] Range:$a[7]-$a[8] E:$a[11]
Bit:$a[12]\n";
       my $name = $a[0]; my $out="Match:$a[3] Range:$a[9]-$a[10]\n
$a[13]";
       $tempout=join("",$tempout,"      $name Num:$alkey{$rank}{$key2}{count}
$out\n");
      }

$subtile{$mkey}{$mnkey}{out}=join("",$subtile{$mkey}{$mnkey}{out},$tempou2,$t
empout);
     }
    }
   }
   for my $mkey (sort keys %subtile ) {
    my $bit=0; my $uskey;
    $tilerslt{$mkey}{num}=scalar(keys(%{$subtile{$mkey}}));
    for my  $mnkey (keys %{$subtile{$mkey}}) {
     if($subtile{$mkey}{$mnkey}{bit}>$bit) {
      $bit=$subtile{$mkey}{$mnkey}{bit};
      $uskey=$mnkey;
     }
    }
    if($tilerslt{$mkey}{num}==1) {
     $tilerslt{$mkey}{sub}=$uskey;
     $tilerslt{$mkey}{detect}=$subtile{$mkey}{$uskey}{detect};
    } else {
#     my $i=0;
     my $i=1;
     for my  $mnkey (keys %{$subtile{$mkey}}) {
      if($mnkey ne $uskey) {
       my $dif = ($subtile{$mkey}{$mnkey}{bit})/$bit;
       if( $dif>0.60) { $i=0; last; }
      }
     }
     $tilerslt{$mkey}{sub}=$uskey;
```

PCT/US2006/022622

```
if($i>0) {
 $tilerslt{$mkey}{detect}=$subtile{$mkey}{$uskey}{detect};
} else {
 my $j=1; my $us2=$uskey;
 my @clip=split(/,/,$subtile{$mkey}{$uskey}{tax});
 for my  $mnkey (keys %{$subtile{$mkey}}) {
  if($subtile{$mkey}{$us2}{tax} ne $subtile{$mkey}{$mnkey}{tax} ) {
   my @b=split(/,/,$subtile{$mkey}{$mnkey}{tax});
   my $i=0;
   while(($i<=$#b && $i<=$#clip)) {
    if($b[$i] ne $clip[$i]) {
     $i=$#b; $j=0;
    }
    $i++;
   }
   last if(!$j);
   if($#b>$#clip) {
    $us2=$mnkey;
    @clip=split(/,/,$subtile{$mkey}{$us2}{tax});
   }
  }
 }
 if($j) {
  $tilerslt{$mkey}{detect}=$subtile{$mkey}{$us2}{detect};
  $tilerslt{$mkey}{sub}=$us2;
 } else {
  $tilerslt{$mkey}{detect}="TAXAMBIG";
 }
}
}
my @clip=split(/,/,$subtile{$mkey}{$tilerslt{$mkey}{sub}}{tax});
$tilerslt{$mkey}{taxcom}=taxname($clip[$#clip]);
}
open(OUT2,join("",">",$Input,".n4rpt"));
open(OUT3,join("",">",$Input,".n4pats"));
open(OUT4,join("",">",$Input,".n4path"));
open(OUT5,join("",">",$Input,".n4tile"));
my $tottile=scalar(keys(%tilerslt));
my %subpatho=(); my %maipatho=();
for my $mkey (sort keys %tilerslt) {
 my $pathogen=$tilerslt{$mkey}{taxcom};
 for my $tarkey (keys %{${$patho}{$mkey}}) {
  if(substr($tarkey,1,-1) <0) {
  if(${${$patho}{$mkey}{$tarkey}{stat} ==0 ) {
   $pathogen=${$patho}{$mkey}{$tarkey}{name};
   last;
  }
  }
 }
 if( exists $subpatho{$pathogen}{num} ) {
  $subpatho{$pathogen}{num}=$subpatho{$pathogen}{num}+1;
  $subpatho{$pathogen}{list}=join(",",$subpatho{$pathogen}{list},$mkey);
  if(index($tilerslt{$mkey}{detect},"SEQUNIQU")+1 &&
index($subpatho{$pathogen}{detect},"TAXUNI")+1 ) {
   $subpatho{$pathogen}{detect}="SEQUNIQU";
  } elsif (index($tilerslt{$mkey}{detect},"TAXUNIQU")+1 &&
index($subpatho{$pathogen}{detect},"SEQUNIQU")+1 ) {
   $subpatho{$pathogen}{detect}="SEQUNIQU";
  }
 } else {
  $subpatho{$pathogen}{list}=$mkey;
  $subpatho{$pathogen}{num}=1;
  $subpatho{$pathogen}{detect}=$tilerslt{$mkey}{detect};
 }
 print OUT5
"$SInput``$tottile`$mkey`$tilerslt{$mkey}{num}`$tilerslt{$mkey}{taxcom}`$tile
rslt{$mkey}{detect}\n";
}
close(OUT5);
$tottile=scalar(keys(%subpatho)); my %maizzz=();
for my $pathogen (sort keys %subpatho) {
```

PCT/US2006/022622

```
print OUT4
"$SInput``$tottile`$pathogen`$subpatho{$pathogen}{num}`$subpatho{$pathogen}{d
etect}`$subpatho{$pathogen}{list}\n";
  my $mkey;
  if((my $l=index($subpatho{$pathogen}{list},","))+1) {
   $mkey=substr($subpatho{$pathogen}{list},0,$l);
  } else {
   $mkey=$subpatho{$pathogen}{list};
  }
  my $pathotax; my $pathogem;
  for my $tarkey (sort {${$patho}{$mkey}{$b}{stat} <=>
${$patho}{$mkey}{$a}{stat}} keys %{${$patho}{$mkey}}) {
   $pathotax=$tarkey;
   $pathogem=${$patho}{$mkey}{$tarkey}{name};
   last;
  }
  if(substr($pathotax,1,-1)>0) {
   if(index($subtile{$mkey}{$tilerslt{$mkey}{sub}}{tax},$pathotax) +1) {
    my @a=split(",",$subtile{$mkey}{$tilerslt{$mkey}{sub}}{tax});
    my $j; for($j=1;$j<=$#a;$j++) { last if($a[$j] == substr($pathotax,1,-
1)); }
    my $list=$a[$j];
    for(my $i=$j+1;$i<=$#a;$i++) { $list=join(",",$list,$a[$i]);}
    $j=$#a-$j;
    if( exists $maipatho{$pathogem}{num} ) {
     $maipatho{$pathogem}{num}=$maipatho{$pathogem}{num}+1;
     if(exists $maipatho{$pathogem}{l}{$j}{$list} ) {

$maipatho{$pathogem}{l}{$j}{$list}=join(",",$maipatho{$pathogem}{l}{$j}{list}
,join(":",$pathogen,$subpatho{$pathogen}{num}));
     } else {

$maipatho{$pathogem}{l}{$j}{$list}=join(":",$pathogen,$subpatho{$pathogen}{nu
m});
     }
    } else {
     $maipatho{$pathogem}{num}=1;

$maipatho{$pathogem}{l}{$j}{$list}=join(":",$pathogen,$subpatho{$pathogen}{nu
m});
    }
   } else {
    if( exists $maizzz{num} ) {
     $maizzz{num}=$maizzz{num}+1;

$maizzz{list}=join(",",$maizzz{list},join(":",$pathogen,$subpatho{$pathogen}{
num}));
    } else {
     $maizzz{num}=1;
     $maizzz{list}=join(":",$pathogen,$subpatho{$pathogen}{num});
    }
   }
  }
 }
 close(OUT4);
 $tottile=scalar(keys(%maipatho));
 for my $pathogen (sort keys %maipatho) {
  my $low; my @clip=();
  for my $mkey (sort {$b <=> $a} keys %{$maipatho{$pathogen}{l}}) {
   for my $mnkey (keys %{$maipatho{$pathogen}{l}{$mkey}}) {
    if(@clip) {
     my @b=split(",",$mnkey);
     my $i=0;
     while(($i<=$#b && $i<=$#clip) && $b[$i] eq $clip[$i]) { $i++; }
     splice(@clip,$i);
    } else {
     @clip=split(",",$mnkey);
    }
   }
  }
  if($#clip == 0) {
```

PCT/US06/022622

```
    print OUT3 "$SInput``$tottile`$pathogen`$maipatho{$pathogen}{num}";
   } else {
    $low=taxname($clip[$#clip]);
    print OUT3 "$SInput``$tottile`$low`$maipatho{$pathogen}{num}";
   }
   for my $mkey (sort keys %{$maipatho{$pathogen}{l}}) {
    $low="";
    print OUT3 "`$mkey;";
    for my $mnkey (keys %{$maipatho{$pathogen}{l}{$mkey}}) {
     $low=join("",$low,",",$maipatho{$pathogen}{l}{$mkey}{$mnkey});
    }
    print OUT3 substr($low,1);
   }
   print OUT3 "\n";
  }
  if(exists $maizzz{num}) {
   print OUT3 "$SInput``$tottile`Non-target`$maizzz{num}`-1;$maizzz{list}\n";
  }
  close(OUT3);
  print OUT2 "$SInput\n";
  print OUT2 "\n Controls\n";
  my $detect="PASS"; my $val="AFFX-TAGIQ-EX"; my $pval="AFFY";
  if(exists $frclist{"AFFX-TAGIQ-EX"}) {
    $val="AFFX-TAGIQ-EX";
  } elsif(exists $frclist{"AFFX-TagIQ-EX"}) {
    $val="AFFX-TagIQ-EX";
  }
  if(exists $frclist{$val}) {
   $detect="FAIL" if($frclist{$val}{per}<20);
   print OUT2 "  $detect $frclist{$val}{per}\% AFFY $frclist{$val}{mat} bases
called of $frclist{$val}{len} with $frclist{$val}{mis} misidentified\n";
  } else {
    print "didn't find $val\n";
  }
  $detect="PASS"; my $tilsuc=1; $val="ATTIM"; $pval=$val;
  if(exists $frclist{$val}) {
   if($frclist{$val}{per}<20) {$detect="FAIL"; $tilsuc=-2;}
   print OUT2 "  $detect $frclist{$val}{per}\% $pval $frclist{$val}{mat} bases
called of $frclist{$val}{len} with $frclist{$val}{mis} misidentified\n";
    print OUT "$val`$SInput`primary`3702`Control`Arabidopsis
thaliana`$tilsuc`0`AY087893`21406667`$frclist{$val}{per}`499`0`0`1`499`426`92
4```1`1`$ltime\n";
  } else {
    print "didn't find $val\n";
  }
  $detect="PASS"; $tilsuc=1; $val="ATNAC1"; $pval=$val;
  if(exists $frclist{$val}) {
   if($frclist{$val}{per}<20){$detect="FAIL"; $tilsuc=-2;};
   print OUT2 "  $detect $frclist{$val}{per}\% $pval $frclist{$val}{mat} bases
called of $frclist{$val}{len} with $frclist{$val}{mis} misidentified\n";
    print OUT "$val`$SInput`primary`3702`Control2`Arabidopsis
thaliana`$tilsuc`0`AC009894`5902358`$frclist{$val}{per}`519`0`0`1`519`82488`8
3006```1`1`$ltime\n";
  } else {
    print "didn't find $pval\n";
  }
  for my $pathogen (sort keys %subpatho ) {
   print OUT2 " $subpatho{$pathogen}{detect} $pathogen in
$subpatho{$pathogen}{num}\n";
    my @a=split(",",$subpatho{$pathogen}{list});
    for my $mkey (sort @a) {
     print OUT2 "  ProSeq $mkey $tilerslt{$mkey}{detect} from
$tilerslt{$mkey}{num} subseq.\n";
     for my $mnkey (sort keys %{$subtile{$mkey}} ) {
      my @b=split("\n",$subtile{$mkey}{$mnkey}{out});
      my @c=split(" ",$b[1]);
      print OUT2 "$b[0] $c[2] $c[3] $c[4] $c[7]\n";
     }
    }
  }
  print OUT2 "DETAILS\n";
```

PCT/US06/22622

```
  for my $pathogen (sort keys %subpatho ) {
   print OUT2 "\n$subpatho{$pathogen}{detect} $pathogen in
$subpatho{$pathogen}{num}\n";
   my @a=split(",",$subpatho{$pathogen}{list});
   for my $mkey (sort @a) {
    print OUT2 "  ProSeq $mkey $tilerslt{$mkey}{detect} from
$tilerslt{$mkey}{num} subseq.\n";
    for my $mnkey (sort keys %{$subtile{$mkey}} ) {
     print OUT2 $subtile{$mkey}{$mnkey}{out};
    }
   }
  }
  close(OUT);
  close(OUT2);
}
sub openconnection
{
 my ($gi) = @_;
 my $username='ncbi';
 my $password='ncbi';
 my $database='bio';
 my $connect="dbi:Oracle:$database";
 my $sqlldrconnection="$username/$password\@$database";
 my $dbh = DBI->connect( $connect,  $username, $password,
                         {
                           RaiseError => 1,
                           AutoCommit => 0
                         }
 )
 || die "Database connection not made: $DBI::errstr";
 return $dbh;
}
sub closeconnection
{
$dbh->disconnect;
 return;
}
sub taxfromgi
{
 my ($gi) = @_;
 my $sth;
 my $taxid=-1;
 my $sql= "select tax_id from gi_taxid_nucl where gi=$gi";
 $sth = $dbh->prepare($sql);
 $sth->execute();
 $sth->bind_columns(undef, \$taxid);
 $sth->fetch();
 my $out=$taxid;
 $sth->finish();
 return($out);
}
sub taxname{
 my ($taxid) = @_;
 my ($sth,$com);
 my $sql= "select name_txt from Taxonomy_names where tax_id=$taxid and
name_class='scientific name'";
 $sth = $dbh->prepare($sql);
 $sth->execute();
 $sth->bind_columns(undef, \$com);
 $sth->fetch();
 $sth->finish();
 return($com);
}
sub taxparents
{
 my ($parentid) = @_;
 my $sth;
 my $sql=""; my $tax;
 my $ch2=join("","",",$parentid,",");
  $sql= "select tax_id AS tax_id from nodes start with tax_id=$parentid
connect by NOCYCLE tax_id= PRIOR parent_tax_id";
```

```
$sth = $dbh->prepare($sql);
$sth->execute();
$sth->bind_columns(undef, \$tax);
$sth->fetch();
while( $sth->fetch() ) {
  $ch2=join("",",",$tax,$ch2);
}
$sth->finish();
return($ch2);
}
```

Obviously, many modifications and variations of the present invention are possible in light of the above teachings. It is therefore to be understood that the claimed invention may be practiced otherwise than as specifically described. Any reference to claim elements in the singular, e.g., using the articles "a," "an," "the," or "said" is not construed as limiting the element to the singular.

References:

1. Whelen, A.C. and Persing, D.H. (1996) The role of nucleic acid amplification and detection in the clinical microbiology laboratory. *Annu Rev Microbiol*, **50**, 349-373.
2. McDonough, E.A., Barrozo, C.P., Russell, K.L. and Metzgar, D. (2005) A multiplex PCR for detection of Mycoplasma pneumoniae, Chlamydophila pneumoniae, Legionella pneumophila, and Bordetella pertussis in clinical specimens. *Mol Cell Probes*, **19**, 314-322.
3. Roth, S.B., Jalava, J., Ruuskanen, O., Ruohola, A. and Nikkari, S. (2004) Use of an oligonucleotide array for laboratory diagnosis of bacteria responsible for acute upper respiratory infections. *J Clin Microbiol*, **42**, 4268-4274.
4. Gardner, S.N., Kuczmarski, T.A., Vitalis, E.A. and Slezak, T.R. (2003) Limitations of TaqMan PCR for detecting divergent viral pathogens illustrated by hepatitis A, B, C, and E viruses and human immunodeficiency virus. *J Clin Microbiol*, **41**, 2417-2427.
5. Ecker, D.J., Sampath, R., Blyn, L.B., Eshoo, M.W., Ivy, C., Ecker, J.A., Libby, B., Samant, V., Sannes-Lowery, K.A., Melton, R.E. *et al.* (2005) Rapid identification and strain-typing of respiratory pathogens for epidemic surveillance. *Proc Natl Acad Sci U S A*, **102**, 8012-8017.
6. Zammatteo, N., Hamels, S., De Longueville, F., Alexandre, I., Gala, J.L., Brasseur, F. and Remacle, J. (2002) New chips for molecular biology and diagnostics. *Biotechnol Annu Rev*, **8**, 85-101.
7. Campbell, C.J. and Ghazal, P. (2004) Molecular signatures for diagnosis of infection: application of microarray technology. *J Appl Microbiol*, **96**, 18-23.
8. Briese, T., Palacios, G., Kokoris, M., Jabado, O., Liu, Z., Renwick, N., Kapoor, V., Casas, I., Pozo, F., Limberger, R. *et al.* (2005) Diagnostic system for rapid and sensitive differential detection of pathogens. *Emerg Infect Dis*, **11**, 310-313.
9. Hacia, J.G. (1999) Resequencing and mutational analysis using oligonucleotide microarrays. *Nat Genet*, **21**, 42-47.
10. Kozal, M.J., Shah, N., Shen, N., Yang, R., Fucini, R., Merigan, T.C., Richman, D.D., Morris, D., Hubbell, E., Chee, M. *et al.* (1996) Extensive polymorphisms observed in HIV-1 clade B protease gene using high-density oligonucleotide arrays. *Nat Med*, **2**, 753-759.
11. Cutler, D.J., Zwick, M.E., Carrasquillo, M.M., Yohn, C.T., Tobin, K.P., Kashuk, C., Mathews, D.J., Shah, N.A., Eichler, E.E., Warrington, J.A. *et al.* (2001) High-throughput variation detection and genotyping using microarrays. *Genome Res*, **11**, 1913-1925.

12. Gingeras, T.R., Ghandour, G., Wang, E., Berno, A., Small, P.M., Drobniewski, F., Alland, D., Desmond, E., Holodniy, M. and Drenkow, J. (1998) Simultaneous genotyping and species identification using hybridization pattern recognition analysis of generic Mycobacterium DNA arrays. *Genome Res*, **8**, 435-448.

13. Lin, B., Vahey, M.T., Thach, D., Stenger, D.A. and Pancrazio, J.J. (2003) Biological threat detection via host gene expression profiling. *Clin Chem*, **49**, 1045-1049.

14. Wilson, W.J., Strout, C.L., DeSantis, T.Z., Stilwell, J.L., Carrano, A.V. and Andersen, G.L. (2002) Sequence-specific identification of 18 pathogenic microorganisms using microarray technology. *Mol Cell Probes*, **16**, 119-127.

15. Wilson, K.H., Wilson, W.J., Radosevich, J.L., DeSantis, T.Z., Viswanathan, V.S., Kuczmarski, T.A. and Andersen, G.L. (2002) High-density microarray of small-subunit ribosomal DNA probes. *Appl Environ Microbiol*, **68**, 2535-2541.

16. Zwick, M.E., McAfee, F., Cutler, D.J., Read, T.D., Ravel, J., Bowman, G.R., Galloway, D.R. and Mateczun, A. (2005) Microarray-based resequencing of multiple Bacillus anthracis isolates. *Genome Biol*, **6**, R10.

17. Wong, C.W., Albert, T.J., Vega, V.B., Norton, J.E., Cutler, D.J., Richmond, T.A., Stanton, L.W., Liu, E.T. and Miller, L.D. (2004) Tracking the evolution of the SARS coronavirus using high-throughput, high-density resequencing arrays. *Genome Res*, **14**, 398-405.

18. Sulaiman, I.M., Liu, X., Frace, M., Sulaiman, N., Olsen-Rasmussen, M., Neuhaus, E., Rota, P.A. and Wohlhueter, R.M. (2006) Evaluation of affymetrix severe acute respiratory syndrome resequencing GeneChips in characterization of the genomes of two strains of coronavirus infecting humans. *Appl Environ Microbiol*, **72**, 207-211.

19. Wang, Z., Daum, L.T., Vora, G.J., Metzgar, D., Walter, E.A., Canas, L.C., Malanoski, A.P., Lin, B. and Stenger, D.A. (2006) Identifying Influenza Viruses with Resequencing Microarrays. *Emerg Infect Dis*, **12**, 638-646.

20. Lin, B., Wang, Z., Vora, G.J., Thornton, J.A., Schnur, J.M., Thach, D.C., Blaney, K.M., Ligler, A.G., Malanoski, A.P., Santiago, J. *et al.* (2006) Broad-spectrum respiratory tract pathogen identification using resequencing DNA microarrays. *Genome Res.* 16:527-535

21. Lin, B., Blaney, K.M., Malanoski, A.P., Ligler, A.G., Schnur, J.M., Metzgar, D., Russell, K.L. and Stenger, D.A. (2006). Naval Research Laboratory.

22. Altschul, S.F., Gish, W., Miller, W., Myers, E.W. and Lipman, D.J. (1990) Basic local alignment search tool. *J Mol Biol*, **215**, 403-410.

23. Davignon, L., Walter, E.A., Mueller, K.M., Barrozo, C.P., Stenger, D.A. and Lin, B. (2005) Use of resequencing oligonucleotide microarrays for identification of Streptococcus pyogenes and associated antibiotic resistance determinants. *J Clin Microbiol*, **43**, 5690-5695.

CLAIMS

What is claimed is:

1. A method comprising:
submitting a plurality of reference sequences in queries to a taxonomic database to produce a plurality of taxonomic results;
wherein the reference sequences are the output of genetic database queries that return a score for each reference sequence; and
reporting a taxonomic identification based on the taxonomic results.

2. The method of claim 1, wherein reporting the taxonomic identification uses one of the following criteria-based steps, the choice of step being the first step wherein the criterion is satisfied:
reporting a single class when the taxonomic results contain only the single class;
reporting the taxonomic result of the reference sequence having the highest score among the plurality of reference sequences when the highest score exceeds the score of the reference sequence having the second highest score by a predetermined score ratio threshold;
reporting a child class when all the taxonomic results contain only the child class and a direct parent class to the child class; and
reporting a common parent class in all other cases.

3. The method of claim 2, wherein the score ratio threshold is about 30%.

4. A method comprising:
performing the method of claim 1 two or more times with different pluralities of reference sequences to generate a plurality of taxonomic identifications;
wherein the inputs to the genetic database queries are subsequences that are correlated to predetermined prototype sequences from target pathogens; and
grouping identical taxonomic identifications with the corresponding prototype sequences.

5. The method of claim 4, further comprising:
reporting a positive result for each of the target pathogens for which one of the taxonomic identifications that corresponds to one of the prototype sequences of the target pathogen is the same as or a child of the target pathogen.

6. The method of claim 5, further comprising:
reporting a final taxonomic identification of:
a single class when the positive results contain only the single class;
a child class when all the positive results contain only the child class and a direct parent class to the child class; or
a common parent class in all other cases.

7. The method of claim 1, wherein the method is computer-implemented.

8. An apparatus comprising:
means for performing the method of claim 7.

9. A method of processing a biological sequence obtained from an assay comprising:
converting base calls located in a predetermined list of positions within the biological sequence to N;
wherein each entry in the predetermined list of positions represents the capability of a substance hybridizing to a microarray used to generate the biological sequence;
wherein the substance is not the nucleic acid of a target pathogen; and
determining the ratio of single nucleotide polymorphisms in the biological sequence relative to a reference sequence.

10. The method of claim 9, wherein the substance is a PCR primer.

11. The method of claim 9, further comprising:
selecting a potential subsequence of an initial length from the biological sequence, when the ratio of single nucleotide polymorphisms is below a SNP threshold; and
calculating the ratio of unique bases within the potential subsequence.

12. The method of claim 11, wherein the SNP threshold is about 20%.

13. The method of claim 11, further comprising:
extending the potential subsequence by one base, when the ratio of unique bases is above a unique-base threshold;
recalculating the ratio of unique bases within the extended potential subsequence;
repeating the extending and recalculating until the ratio of unique bases is below the unique-base threshold;
removing trailing N base calls from the extended potential subsequence to form a query subsequence; and
calculating the ratio of unique bases within the query subsequence.

14. The method of claim 13, wherein the unique-base threshold is about 40%.

15. The method of claim 13, wherein repeating the extending and recalculating is stopped when the last 21 locations of the extended potential subsequence have fewer than 4 unique base calls.

16. The method of claim 13, further comprising:
submitting the query subsequence to a genetic database in a query to identify the biological sequence, when the length of the query subsequence and the ratio of unique bases within the query subsequence meet predetermined requirements.

17. The method of claim 16, wherein the requirements include:

the query subsequence contains at least 7 contiguous unique base calls; and

the length of the query subsequence is more than 100 bases, the length of the query subsequence is from 30 to 100 bases and ratio of unique bases within the query subsequence is at least about (((the length of the query subsequence -30)*0.2857 + 70) %, or the length of the query subsequence is less than 30 bases and ratio of unique bases within the query subsequence is at least about 95%.

18. The method of claim 9, wherein the method is computer-implemented.

19. An apparatus comprising:

means for performing the method of claim 18.