(19)中华人民共和国国家知识产权局

(12)发明专利

(10)授权公告号 CN 101390101 B
(45)授权公告日 2012.05.23

(21)申请号 200780005575.0
(22)申请日 2007.02.15
(30)优先权数据
60/774,354 2006.02.16 US
(85)PCT申请进入国家阶段日
2008.08.15
(86)PCT申请的申请数据
PCT/US2007/004187 2007.02.15
(87)PCT申请的公布数据
WO2007/098049 EN 2007.10.25
(73)专利权人 454生命科学公司
地址 美国康涅狄格州
(72)发明人 陈怡儒 K·麦达德 J·辛普森
(74)专利代理机构 中国专利代理(香港)有限公司 72001
代理人 刘杰 陈景峻
(51)Int.Cl.
G06F 19/22(2011.01)

(56)对比文件
WO 2005040425 A2,2005.05.06, 全文.
US 2004197845 A1,2004.10.07, 全文.
CN 1662662 A,2005.08.31, 全文.

审查员 刘欢

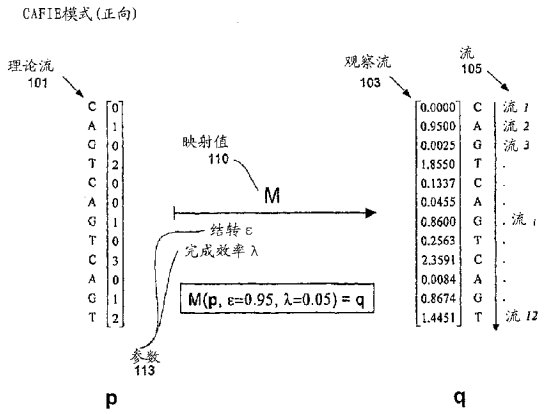
权利要求书 2 页 说明书 45 页 附图 9 页

(54)发明名称

用于校正核酸序列数据中的引物延伸误差的系统和方法

(57)摘要

描述一种用于校正与从模板分子的基本相同副本种群中产生的相位同步序列数据相关联的误差的方法的实施例,包括:(a)检测响应于测序反应中一个或多个核苷酸并入而产生的信号;(b)生成用于信号的值;以及(c)利用第一参数和第二结转参数校正用于相位同步误差的值。



CN 101390101 B

1. 一种用于校正与模板分子的相同副本的种群中产生的相位同步序列数据相关联的误差的方法，包括：

(a) 检测响应于测序反应中一个或多个核苷酸的并入而产生的信号；

(b) 生成表示所述信号的信号值；以及

(c) 利用完成效率参数和结转参数校正相位同步误差的信号值。

2. 如权利要求 1 所述的方法，进一步包括：

(d) 对于每一个模板分子的序列位置重复步骤 (a)-(c)。

3. 如权利要求 2 所述的方法，进一步包括：

(e) 将每个校正后的值并入到模板分子的序列的流程图中。

4. 如权利要求 2 所述的方法，其中：

该相位同步误差包括不完全延伸分量和结转分量，两者都实质上被视为用于每个模板分子的序列位置的常量，其中完成效率参数代表不完全延伸分量，结转参数代表结转分量。

5. 如权利要求 2 所述的方法，其中：

该相位同步误差包括结转分量，其实质上被视为用于每个模板分子的序列位置的常量，其中该结转参数代表结转分量。

6. 如权利要求 1 所述的方法，其中：

该信号包括响应于一个或多个核苷酸并入而发出的光。

7. 如权利要求 6 所述的方法，其中：

该光包括来自测序反应的化学发光。

8. 如权利要求 7 所述的方法，其中：

该测序反应包括焦磷酸盐测序反应。

9. 如权利要求 6 所述的方法，其中：

该光包括来自测序反应的荧光。

10. 如权利要求 9 所述的方法，其中：

该测序反应包括使用可逆终止子的测序反应。

11. 如权利要求 1 所述的方法，其中：

信号值表示大量并入的核苷酸。

12. 一种用于校正与模板分子的相同的副本种群中产生的相位同步序列数据相关联的误差的方法，包括：

(a) 检测响应于测序反应中一个或多个核苷酸的并入而产生的信号；

(b) 生成表示所述信号的信号值；

(c) 并入该信号值到与模板分子序列关联的流程图；

(d) 对每一个模板分子的序列位置重复步骤 (a)-(c)；

(e) 利用完成效率参数和结转参数校正所述流程图中的相位同步误差的每个值；以及

(f) 利用校正后的值生成校正的流程图。

13. 如权利要求 12 所述的方法，进一步包括：

(g) 利用步骤 (e) 中前一个迭代获得的校正值迭代重复步骤 (e)-(f)，其中一些或所有这些校正值随着每次迭代在质量上有所改进。

14. 如权利要求 12 所述的方法，其中：

该相位同步误差包括不完全延伸分量和结转分量，两者都实质上被视为用于每个模板分子的序列位置的常量，其中完成效率参数代表不完全延伸分量以及结转参数代表结转分量。

15. 如权利要求 12 所述的方法，其中：

该相位同步误差包括结转分量，其实质上被视为用于每个模板分子的序列位置的常量，其中该结转参数代表结转分量。

16. 如权利要求 12 所述的方法，其中：

该信号包括响应于一个或多个核苷酸并入而发出的光。

17. 如权利要求 16 所述的方法，其中：

该光包括来自测序反应的化学发光。

18. 如权利要求 17 所述的方法，其中：

该测序反应包括焦磷酸盐测序反应。

19. 如权利要求 16 所述的方法，其中：

该光包括来自测序反应的荧光。

20. 如权利要求 19 所述的方法，其中：

该测序反应包括使用可逆终止子的测序反应。

21. 如权利要求 12 所述的方法，其中：

信号值表示大量并入的核苷酸。

22. 一种用于校正与模板分子的相同的副本种群中产生的相位同步序列数据相关联的误差的方法，包括：

(a) 检测响应于测序反应中一个或多个核苷酸的并入而产生的信号；

(b) 生成表示所述信号的信号值；

(c) 并入该信号值到与模板分子序列关联的流程图中；

(d) 对每一个模板分子的序列位置重复步骤 (a)-(c)；

(e) 将该流程图分成多个子集，其中每个子集包括一个或多个模板分子的序列位置；

(f) 估计每个子集中完成效率参数和结转参数的同步误差；

(g) 利用每个各自子集中的完成效率参数和结转参数的同步误差估计来校正用于相位同步误差的每个子集中的每个值；以及

(h) 利用校正后的值将该校正子集组合成校正的流程图。

23. 如权利要求 22 所述的方法，其中：

该相位同步误差包括不完全延伸分量和结转分量，所述不完全延伸分量和结转分量在模板分子的多个序列位置的上方波动，其中完成效率参数代表不完全延伸分量以及结转参数代表结转分量。

24. 如权利要求 22 所述的方法，其中：

该相位同步误差包括结转分量，所述结转分量在模板分子的多个序列位置的上方波动，其中该结转参数代表结转分量。

用于校正核酸序列数据中的引物延伸误差的系统和方法

[0001] 相关申请

[0002] 本申请涉及并且要求2006年2月16日提交的，标题为“用于校正核酸序列数据中的引物(primer)延伸误差的系统和方法”的美国临时专利申请系列No.60/774,354的优先权，因此其全部内容在此被引入作为参考用于所有目的。

技术领域

[0003] 本发明涉及分子生物学领域。更特别地，本发明涉及校正由一般被称为“合成测序”(SBS)(sequencing-by-synthesis)的技术生成的核酸序列数据中的误差。

背景技术

[0004] 合成测序(SBS)一般是指用于确定核酸样本中一个或多个核苷酸的特征或序列构成的方法，其中该方法包括将单链的多核苷酸分子补体逐步合成为模板核酸分子，其中确定了核苷酸序列组成。例如，SBS技术一般是通过在相应的序列位置上添加单一的核酸(也称为核苷酸)核素到新多核苷酸分子补体以形成模板分子的核酸核素来操作。对新分子添加核酸核素一般是利用各种本领域公知的方法进行检测，该方法包括，但不限定于称作为焦测序或荧光检测方法的方法，诸如那些采用了可逆终止子的方法。一般地，重复操作过程直到完成(即，表示了所有的序列位置)或合成了模板想要的序列长度补体(complementary)。SBS技术的一些例子在US专利No.6,274,320中描述，因此其全部内容在此被引入作为参考用于所有用途；以及美国专利申请系列号为No.10/788,529；09/814,338；10/299,180；10/222,298；10/222,592，因此它们中的每一个的全部内容在此被引入作为参考用于所有目的。

[0005] 在SBS的一些具体实施方式中，低核苷酸引物被设计成退火到样品模板分子的预定的补充位置。引物/模板合成物在核酸聚合酶的存在下可能表现为核苷酸核素(nucleotide specie)。如果核苷酸核素是对核酸核素的补充，那么聚合酶将用核苷酸核素延伸引物，其中核酸核素对应于紧邻低核苷酸引物的3’末端的样品模板分子上的序列位置。或者，在一些实施例中，引物/模板合成物被立即呈现成多个感兴趣的核苷酸核素(代表性的A,G,C和T)，并且核苷酸核素被合并，该核苷酸核素是在紧邻低核苷酸引物的3’末端的样品模板分子上的对应的序列位置的补充。在任何一个上述实施例中，核苷酸核素可以以化学方法被封闭(比如在3’-O位置)以防止进一步延伸，并且需要在第二轮合成前解除封闭。如上所述，核苷酸核素的合并可以通过本领域中已知的各种方法检测，如通过检测焦磷酸盐(PPi)的释放(在美国专利No.6,210,891；6,258,568；和6,828,100中描述的范例，因此每个范例的全部内容在此被引入作为参考用于所有目的)，或经由绑定到核苷酸的可检测的标签。可检测标签的一些例子包括但不限于质量标记和荧光或化学发光的标签。在一些典型实施例中，未合并的核苷酸会被移除，例如被洗涤。在可检测的标签被使用的实施例中，它们通常会在接下来的合成循环之前被失活(例如，通过化学的分裂或光致退色)。在模板/聚合酶合成物的下一个序列位置可以用另一个核苷酸核素或如上所述的多

个感兴趣的核苷酸核素查询。核苷酸添加、引物延伸、信号获取、和洗涤的重复循环导致模板链的核苷酸序列的确定。

[0006] 在SBS的典型的实施例中，很多或大量的基本上相同的模板分子（例如10^3，10^4，10^5，10^6或10^7分子）以任何一种测序反应被同时分析，以便实现对于可靠的检测足够强的信号。为了低信噪比，需要在大量的给定反应中与基本上所有模板分子有关的被称作为未成熟分子的"均匀延伸"。如在此使用的术语"均匀延伸"，一般指的是延伸反应的关系或相位，在延伸反应中上述的基本上相同的每一个模板分子在反应中均匀地执行相同的步骤。例如，每个与多个模板分子有关的延伸反应可以被描述为当它们在相同的序列位置为每个相关的模板分子执行相同的反应步骤时处于同相或彼此相位同步。

[0007] 然而那些本领域中的普通技术人员将要理解，在每个群体中的小部分模板分子与群体中剩余的模板分子错过或失去了相位的同步性（也就是说，与一部分模板分子有关的反应或者超过，或者落后对群体进行的测序反应中的其它模板分子（在Ronaghi，M.Pyrosequencing sheds light onDNA sequencing Genome Res.11,3-11(2001)中描述了一些例子，其全部内容在此被引入作为参考用于所有目的）。例如，将一个或多个核苷酸核素适当地合并成为一个或多个未成熟的分子用于延伸该序列一个位置的反应的失败，导致随后的每个反应处于在群体剩余的序列位置后的和与群体剩余的序列位置不同相的序列位置。这个效果在此称为"不完全的延伸"(IE)。可替换的，通过将一个或多个核苷酸核素合并到领先于和与群体其余的序列位置不同相的序列位置中而不适当延伸的未成熟分子在此被称为"结转"(CF)(carry forward)。CF和IE的综合效应在此被称为CAFIE。

[0008] 对于不完全延伸的问题，也许存在有一些可能的机制，其有助于可能单独出现或在一些组合中出现的IE。可能的有助于IE机制的一个例子可以包括，缺少表现为模板/聚合酶合成物的子集的核苷酸核素。可能的有助于IE机制的另一个例子可以包括，聚合酶分子的子集没有成功的合并核苷酸核素，该核苷酸核素被适当地呈现用于合并成未成熟的分子。可能的有助于IE机制的另一个例子可以包括在模板/聚合酶合成物中缺少聚合酶活动。

[0009] 至少部分地用于SBS方法中的IE误差的又一个可能考虑机制的例子可以包括，由Metzger(Genome Res.2005Dec;15(12):1767-76，其全部内容在此引入作为参考用于所有目的）所述的所谓的循环可逆终止(CRT)。在CRT中，核苷酸核素具有修改的3′-O组（通常被称为帽，保护组，或终止子），其防止在单个核苷酸核素合并之后未成熟分子的进一步延伸。这些保护组通过各种方法的一种被设计成可移动的，其中一种方法包括化学处理或光照处理。一旦3′-O位置去保护（以及3′-OH组创建），未成熟的分子可能通过另一个核苷酸核素被延伸。然而，当少许未成熟的分子保持受保护的状态时，由于未完成的去保护效果（不完全的去保护）将会出现相位的异步。在随后的循环中，这部分保持受保护的状态的未成熟的分子将不会被延伸，因而将落后于群体剩余的序列位置并且与群体剩余的序列位置不同相。然而，随后的去保护步骤可以成功地删除已经预先不正确保留的至少一些保护组，使延伸恢复，并且从未成熟的分子中创建信号并且继续与剩余的群体非相位同步。那些本领域中的普通技术人员将会理解，可能存在其它有助于IE的因素，因而不局限于上面提供的例子。

[0010] 目前描述的本发明实施例的系统和方法旨在校正可能由任何这种单独的或组合

的原因或机制而产生的 IE 误差。例如，由不完全的去保护和随后成功的去保护的结合所引起的 IE 误差的校正是本发明的一个目的。

[0011] 对于 CF 的问题，也许存在有一些可能的机制，其有助于可能单独出现或在一些组合中出现的 CF。例如，一个可能的机制可以包括从前一循环中剩余的过量的核苷酸核素。这种情况是可能出现的，这是因为在循环的未端执行的洗涤协议将从循环中删除大多数的但都是不必要的核苷酸核素。在本例子中结果可能包括存在于 " G " 核苷酸核素循环中的 " A " 核苷酸核素的小部分，如果互补的 " T " 核苷酸核素存在于模板分子中的对应的序列位置则延伸未成熟分子的小部分。引起结转效果的可能机制的另一个例子可以包括聚合酶误差，诸如不适当的合并核苷酸核素成为与模板分子上的核苷酸核素不互补的未成熟的分子。

[0012] 至少部分地用于 SBS 方法中的 CF 的又一个可能考虑机制的例子包括，如由 Metzger(Genome Res.2005Dec ;15(12) :1767-76，上述的合并作为参考）评述的循环可逆终止 (CRT)。在本例子中，如上所述对于 IE 也许采用对 3′ -O 受保护的核苷酸核素的准备，其中核苷酸分子的一些部分将缺少保护组，或已经丢失该保护组。保护组的遗失也可能出现在打算的去保护步骤之前的测序处理期间。任何这种去保护组的缺乏将导致一些未成熟的分子将由每次超过一个的核苷酸核素延伸。这种不适当的对一小部分未成熟分子的多个延伸导致它们在序列位置中正向移动并且与群体剩余的序列位置不同相。因而，未被保护的核苷酸和 / 或过早去保护的核苷酸，可以至少部分地有助于包含 CRT 的 SBS 方法中的 CF。

[0013] 目前描述的本发明实施例的系统和方法旨在校正可能由任何这种单独的或组合的原因或机制而产生的 CF 误差。例如，校正由于缺乏保护组而出现的 CF 误差是本发明的一个目的。

[0014] 进一步，目前描述的本发明实施例的系统和方法旨在校正 IE 误差和 CF 误差两者，其中两种类型的错误可能出现在用于相同测序反应中的群体的一些组合中。例如，每个 IE 和 CF 可能由如上所述的单独的或组合的原因或机制产生。

[0015] 那些普通技术人员将要理解，对于 IE 和 CF 两者误差的可能性也许出现在延伸反应期间每个序列位置，因而可能在最后所得到的序列数据中具有明显的累积效应。例如，该效果对于一系列测序反应的末尾可能变得尤其显著，其有时也被称为 " 试验 " (run) 或 " 测序试验 " (sequencingrun)。进一步，IE 和 CF 效果可以规定可以使用 SBS 方法可靠地测序的模板分子长度的上限（有时被称为 " 读取长度 "），这是因为序列数据的质量随着读取长度增加而减少。

[0016] 例如，SBS 的一个方法可以在典型试验中生成包括超过 2 千 5 百万个序列位置的具有 20 或更多的被称为 " Phred " 的质量得分 (20 的 Phred 质量得分推断序列数据被预测为具有 99％或更高的准确性）的序列数据。当对于 SBS 方法具有 Phred20 质量的整个测序吞吐量显著地高于由如采用毛细管电泳技术的 Sanger 测序方法的现有技术中众所周知的那些生成的序列数据的吞吐量时，则对于 SBS 方法目前以基本上较短的读取长度为代价 (Margulies 等，2005，Nature437 :376-80，因此其全部内容在此被引入作为参考用于所有目的）。因而通过避免或校正由 IE 和 CF 误差产生的序列数据的降级增加读取长度的上限将导致 SBS 方法的整个测序吞吐量的增加。

[0017] 因此，所希望的是提供旨在校正在通过核酸测序的合成测序方法产生的序列数据

中的 IE 和 / 或 CF 误差的系统和方法。

[0018] 多个参考文件援引在此，它们的整个公开在此引入作为参考用于所有的目的。进一步，无论上面如何表征，这些参考中没有一个被认为是对在此请求保护主题的本发明的现有技术。

发明内容

[0019] 本发明的实施例涉及确定核酸序列。更具体的说，本发明实施例涉及用于校正在由 SBS 对核酸进行排序期间获得的数据中的误差的方法和系统。

[0020] 描述一种用于校正与从模板分子的基本相同副本的种群中产生的序列数据的相位同步相关联的误差的方法的实施例，包括 :(a) 检测响应于测序反应中一个或多个核苷酸并入产生的信号 ;(b) 生成用于信号的值 ;以及 (c) 利用第一参数和第二参数校正用于相位同步误差的值。

[0021] 在一些执行过程中，为模板分子的每个序列位置重复 (a)-(c) 步骤，并且每个被校正的值可以被合并到可以包括流程图表示的模板分子的表示中。

[0022] 除此之外，一种用于校正与从模板分子的基本相同副本的种群中产生的序列数据的相位同步相关联的误差的方法的实施例，包括 :(a) 检测响应于测序反应中一个或多个核苷酸并入产生的信号 ;(b) 生成用于信号的值 ;(c) 并入该值到与模板分子序列关联的表示中 ;(d) 对每一个模板分子的序列位置重复步骤 (a)-(c) ;(e) 利用第一参数和第二参数校正表示中的相位同步误差的每个值 ;以及 (f) 利用校正值生成校正表示。

[0023] 此外，一种用于校正与从模板分子的基本相同副本的种群中产生的序列数据的相位同步相关联的误差的方法的实施例，包括 :(a) 检测响应于测序反应中一个或多个核苷酸并入产生的信号 ;(b) 生成用于信号的值 ;(c) 并入该值到与模板分子序列关联的表示中 ;(d) 对每一个模板分子的序列位置重复步骤 (a)-(c) ;(e) 将该表示分成多个子集，其中每个子集包括模板分子的一个或多个序列位置 ;(f) 估计每个子集中第一参数和第二参数的同步误差 ;(g) 利用每个各自子集的第一参数和第二参数的同步误差估计校正每个子集中的用于相位同步误差的每个值 ;以及 (h) 利用校正值将该校正子集组合成校正表示。

[0024] 另外地，描述了一个用于校正与从模板分子的基本相同副本的种群中产生的序列数据的相位同步相关联的误差的系统的实施例，包括具有存储在其上用于执行的程序代码的计算机，其执行的方法包括 :(a) 响应测序反应中一个或多个核苷酸并入生成检测的信号值 ;以及 (b) 利用第一参数和第二参数校正相位同步误差值。

[0025] 更进一步地，描述一个用于校正与从模板分子的基本相同副本的种群中产生的序列数据的相位同步相关联的误差的系统的实施例，包括具有存储在其上用于执行的程序代码的计算机，其执行的方法包括 :(a) 响应测序反应中一个或多个核苷酸并入生成检测的信号值 ;(b) 合并该值到与模板分子的序列关联的表示中 ;(c) 对模板分子的每个序列位置重复步骤 (a)-(b) ;(d) 利用第一参数和第二参数校正表示中的相位同步误差的每个值 ;以及 (e) 利用校正值生成校正表示。

[0026] 此外，描述一个用于校正与从模板分子的基本相同副本的种群中产生的序列数据的相位同步相关联的误差的系统的实施例，包括具有存储在其上用于执行的程序代码的计算机，该程序代码执行的方法包括 :(a) 响应测序反应中一个或多个核苷酸并入生成检测

的信号值；(b) 合并该值到与模板分子的序列关联的表示中；(c) 对模板分子的每个序列位置重复步骤 (a)-(c)；(d) 将该表示分成多个子集，其中每个子集包括模板分子的一个或多个序列位置；(e) 估计每个子集中第一参数和第二参数的同步误差；(f) 利用每个各自子集中的第一参数和第二参数的同步误差估计来校正每个子集中的用于相位同步误差的每个值；以及 (g) 利用校正值将该校正子集组合成校正表示。

[0027] 由本发明的实施例实现的优点包括但并不局限于：(a) 序列数据质量得到增加，从而得到实现想要级别的精确度的合意序列所需的更少的序列覆盖深度；(b) 有用的序列读取长度被延伸，也就是说更高质量序列数据可以从单次试验中获取；(c) 因为有用的序列读取长度被延伸，需要更少的试验就能实现给定深度的序列覆盖；(d) 因为有用的序列读取长度被延伸，需要更少的序列来组合跨越给定区域的序列 contig；和 (e) 结果增加的读取长度促进交叠读取的组合，尤其是在重复的序列区域中。

附图说明

[0028] 结合附图，以上及其更多的特征将会被通过下列详细的描述更清晰的理解。在附图中，相同的参考数字代表相同的结构、元件或方法步骤并且参考数字最左边的阿拉伯数字表示参考元件初次出现的附图的数字（例如，元件 160 最早出现在图 1 中）。然而，所有这些协定，是典型性的或解释说明性的，而不是限制性的。

[0029] 图 1 是用于将“完美的”理论流程图转换为“不清晰的”(dirty) 观察流程图的数学模型的一个实施例的简化图形表示；

[0030] 图 2 是图 1 中映射模型反转的一个实施例的简化图形表示；

[0031] 图 3a 是用于正向及反转包含图 1 和图 2 中映射模型的矩阵计算的模型的一个实施例的简化图形表示；

[0032] 图 3b 是采用图 3a 正向模型的正向矩阵计算的一个实施例的简化图形表示；

[0033] 图 4a 是采用图 3a 反转模型的反转矩阵计算的一个实施例的简化图形表示；

[0034] 图 4b 是利用不同级别的使用图 3a 和 4a 的反转模型的迭代 (iterative) 校正而得到的结果的一个实施例的简化图形表示；

[0035] 图 5 是当前描述发明的 CAFIE 误差校正方法的结果的一个实施例的简化图形表示；

[0036] 图 6 是基本相同模板分子种群的样本上分配参数值的一个实施例的简化图形表示；以及

[0037] 图 7 是仅仅 IE 校正的效果和 CAFIE 校正效果的一个实施例的简化图形表示。

具体实施方式

[0038] 当前描述发明的实施例至少部分基于理论上或“完美”的流程图可以通过 IE 和 CF 的数学模型被转化为现实生活中可观察的”不清晰”流程图的发现。在这里使用的术语“流程图”一般泛指，从测序试验中产生的测序数据的表示，其可以例如包括测序数据的图形表示。例如，完美的或者理论上的流程图表示从测序试验中产生的数据，该测序试验根据上述的 CAFIE 机制没有误差，或有其他类型的背景误差。沿着相同的线不清晰的或可观察的流程图表示从测序试验中生成的数据，其中该测序试验包括 CAFIE 和背景误差因素。在本例

中，一些或所有误差因素可以被准确近似和运用到完美的流程图模型中，以提供从实际的测序试验中得到的真实数据的表示。

[0039] 重要的是，当前描述的发明也至少部分地基于上面描述的数学模型的反转可以从一个不清晰的可观察流程图中来近似完美理论流程图的发现。然而，继续上面近似误差的例子可以应用到在可观察的流程图中表示的实际测序数据中，从而形成具有可移除的所有或基本上所有的误差因素的实际序列数据的完美或基本上完美的理论流程图表示。

[0040] 本领域的普通技术人员将会理解，准确移除数据中的误差会提供给所述数据更有效率和准确的解释。因此，例如，从测序试验中产生的数据中移除误差会导致在从序列试验中产生的序列中识别每个核酸核素的调用的更准确的产品，以及更高质量的序列信息。

[0041] 当前描述发明的一些实施例包括用于分析在测序设备上的 SBS 测序试验产生的数据的系统和方法。SBS 设备和方法的一些例子可以使用可被称为基于焦磷酸盐的测序方法，其可能例如包含一个或多个检测装置例如，电荷耦合装置 (CCD) 照相机，微射流室，样品带盒托座，或泵和流量阀。以基于焦磷酸盐的测序为例，设备的实施例可以使用化学发光作为检测方法，该化学发光对于焦磷酸盐测序产生内在的低分贝的背景噪音。在本例中，用于测序的样品带盒托座可以包括所谓的“铬尖晶石托盘”，它是由一个酸腐蚀的纤维光学面板形成的，以产生成百上千的非常细小的井 (well)，每个井能够容纳大量的基本上相同的模板分子。在一些实施例中，基本上相同的模板分子的每个群体可以被布置在固体衬底上，如珠子，每个珠子可以布置在所述井之一中。继续本例子，设备可以包括 :试剂传送元件，用来提供液体试剂到铬尖晶石托盘支架上 ;和 CCD 类的检测装置，能够收集从铬尖晶石托盘上每个井中射出的光子。更多的用于执行 SBS 类测序和焦磷酸盐测序的设备和方法例子在美国专利申请序列号 No.10/767,779 ;11/195,254 中有描述，因此这两个文件的全部内容在此被引入作为参考用于所有目的。

[0042] 更进一步的，当前描述发明的实施例的系统和方法可以实施为存储在计算机可读介质上，以用于在计算机系统上执行。例如，下面详细描述了一些实施例，以处理和校正使用在计算机系统上可实现的 SBS 系统和方法检测的信号中的误差。

[0043] 计算机可以包括任何类型的计算机平台，如工作站、个人计算机、服务器、或任何当前或未来的计算机。计算机通常包括已知的部件如处理器、操作系统、系统内存、存储器存储装置、输入输出控制器、输入输出装置、和显示装置。相关领域中的那些普通技术人员应当理解，计算机可能会有很多可能的配置和部件，并也可能包括高速缓冲存储器、数据备份单元、和很多其他装置。

[0044] 显示装置可以包括提供可视信息的显示装置，此信息一般可以被逻辑地和 / 或物理性地组织为像素阵列。也可以包括界面控制器，界面控制器可以包括任何类型的用于提供输入输出界面的已知或未来的软件程序。例如，界面可以包括一般被定义为“图形用户界面”(一般称作 GUI) 的界面，图形用户界面提供给用户一个或多个图形表示。界面通常能够接受用户使用本领域中那些普通技术人员已知的选择或输入装置进行的输入。

[0045] 在相同或可替换的实施例中，计算机上的应用程序可以采用包括被称为“命令行界面”(经常称为 CLI) 的界面。在应用程序和用户之间，CLI 通常提供基于文本的交互。典型的，命令行界面通过显示装置显示输出和接收输入作为文本行。举个例子，一些实现方法可以包括所谓的“壳”，如本领域中的那些普通技术人员已知的 UnixShells，或

MicrosoftWindows Powershell，其采用面向对象类型的编程体系结构例如 Microsoft.NET framework。

[0046] 本领域普通技术人员将要理解，界面可以包括一个或多个 GUI，CLI 或它们的组合。

[0047] 处理器可以包括商业上可用的处理器如英特公司生产的 Itanium® 或奔腾处理器，Sun Microsystems 公司生产的 SPARC® 处理器，AMD 公司生产的 Athalon™ 或 Opteron™ 处理器，或它也可以是一个正在或将要变成可以使用的其他处理器。处理器的一些实施例也可以包括所谓的多核处理器和 / 或能够在单核或多核配置中采用并行处理技术。例如，多核结构一般包括两个或多个处理器“执行核”。在当前例子中，每个执行核可以以作为能够并行执行多个线程的独立处理器执行。另外，本领域中的那些普通技术人员将会理解，处理器可以被设置成一般所谓的 32 或 64 位结构，或其他现在已知或将来会被开发的体系结构。

[0048] 处理器一般运行操作系统，例如，操作系统可以是微软公司的 WINDOWS ® 型操作系统（如 Windows® XP 或 Windows Vista®）；苹果电脑公司的 Mac OS X 操作系统（如 7.5Mac OS X v10.4“Tiger”或 7.6MacOS X v10.5“Leopard”操作系统）；Unix 或 Linux 型操作系统可以从很多卖主或所谓的公开渠道得到；其它或未来的操作系统；或它们的一些组合。操作系统通过公知方式与固件和硬件接口，并且帮助处理器调整和执行各种能以各种编程语言写入的计算机程序的功能。操作系统，一般和处理器协作，协调和执行计算机其他部件的功能。操作系统同样也提供进度表、输入 - 输出控制、文件和数据管理、存储管理、以及通信控制及相关服务，所有的都依照已知的技术。

[0049] 系统存储器可以包括任何类型的已知或未来的存储器存储设备。例子包括任何一般可以获得的随机存取存储器 (RAM)，磁介质例如驻存硬盘或磁带，光学介质例如读和写光盘，或其他存储器存储设备。存储器存储设备可以包括任何类型已知的或未来的设备，包括光盘驱动、磁带驱动、可移动硬盘驱动、USB 或闪存、或磁盘设备。这种类型的存储器存储设备一般读自和 / 或写入到程序存储介质中（未示出）例如，分别为光盘、磁带、可移动硬盘、USB 或闪存或软盘。这些程序存储介质中的任何一个或其他现在使用的或也许以后会开发的可以视为计算机程序产品。如所希望的，这些程序存储介质一般存储计算机软件程序和 / 或数据。计算机软件程序，同样称为计算机控制逻辑，一般被存储在系统存储器中和 / 或用于连接存储器存储设备的程序存储设备中。

[0050] 在一些实施例中，计算机程序产品被描述为包括计算机可用介质，该计算机可用介质具有存储在其中的控制逻辑（计算机软件程序，包括程序代码）。该控制逻辑，当由处理器执行时，使得处理器执行这里所述的功能。在其他实施例中，一些功能主要由使用例如硬件状态机的硬件实行。执行硬件状态机以便执行这里所述的功能对于本领域相关技术人员来说将是显而易见的。

[0051] 输入 - 输出控制器可以包括任何类型的已知的用于接收和处理来自用户信息的设备，该用户无论是人还是机器，无论是本地的还是远程的。这样的设备包括，例如调制解调器卡、无线卡、网络接口卡、声卡、或用于任何类型已知输入设备的其他类型的控制器。输出控制器可以包括用于向用户显示信息的任何类型的已知显示设备的控制器，该用户无论是人还是机器，无论是本地还是远程。在当前描述的实施例中，计算机的功能元件通过系统总线彼此相互通信。计算机的一些实施例可以利用网络或其他类型的远程通信与一些功能

性的元件互相通信。

[0052] 正如相关领域的技术人员显然得知的，工具控制和 / 或数据处理应用，如果用软件执行，则可以被载入并从系统内存和 / 或存储器存储设备中执行。所有或部分工具控制和 / 或数据处理应用同样可以驻留在只读存储器中或类似于存储器存储设备的设备中，这样的设备不要求工具控制和 / 或数据处理应用通过输入 - 输出控制器被首先加载。相关领域技术人员将要理解的是，工具控制和 / 或数据处理应用或它们的一部分可以由处理器以公知的方式被载入到系统内存中，或高速缓冲存储器中，或二者中，作为执行的优势。

[0053] 同样，计算机可以包括存储在系统内存中的一个或多个库文件、试验数据文件、以及互联网客户。例如，试验数据可以包括与一个或多个试验或分析相关的数据比如检测信号值，或其他与一个或多个 SBS 试验或处理相关联的值。此外，互联网客户可以包括能利用网络访问另一个计算机上的远程服务的应用，例如可以包括所谓的“Web 浏览器”。在当前例子中一些通常使用的 web 浏览器包括从 Netscape Communications 公司获得的 Netscape® 8.1.2，从 Microsoft 公司获得的 Microsoft® InternetExplorer7，从 Mozilla 公司获得的 Mozilla Firefox® 2，从 Apple 计算机公司获得的 Safari1.2，或技术人员现在已知的或将来可能开发的其他类型的 web 浏览器。此外，在同一实施例或其他实施例中，互联网客户可以包括（或可能成为一个元件）专用软件应用程序，该专用软件应用程序使得能经由网络例如 SBS 应用所使用的数据处理应用来访问远程信息。

[0054] 网络可以包括一个或多个本领域普通技术人员所熟知的不同类型的网络。例如，网络可以包括使用通常所说适用于通信的 TCP/IP 协议的局域或广域网。网络可以包括具有彼此互连的计算机网络的世界范围系统的网络，其一般称之为互联网，或还可以包括各种内联网设施。相关领域普通技术人员将可以理解网络互连环境中的一些用户可以最好使用一般所说的“防火墙”（有时候也称为包过滤器，或边界保护设备）来控制与硬件和 / 或软件系统的信息交换。例如，防火墙可以包括硬件或软件元件或其中的一些组合，并且一般被设计为用户可以加强安全规则，例如用于即时的网络管理等。

[0055] SBS 实施例的例子一般使用添加到如上所述的模板分子中的核苷酸核素的连续或迭代的循环。这里这些循环也被称为“流”(flow)。例如，在每个流中，或者四个核苷核核素中的一个，A，G，C 或 T 被表示（例如，对于焦磷酸盐 (PPi) 测序法）；或者四个所有的核苷酸核素一起被表示为模板 - 聚合酶合成物（例如，用于使用与每个核苷酸核素相关的不同标签的测序方法）。继续当前例子，一个流可以包括在序列位置处模板分子中的核苷酸核素的核苷酸核素补体，该序列位置紧邻于正被合成的未成熟分子的 3’ 末端，其中核苷酸核素被合并到未成熟分子中。在当前例子中，核苷酸核素的合并可以以光信号（如，光信号可以例如包括从发光或荧光处理中产生的光）或其他信号如质量标签的形式被检测到。在每个核苷酸核素的流的迭代中，洗涤方法是通过去除没有被并入的剩余的核苷酸核素和试剂来实施的。在完成洗涤阶段之后，下一个流的迭代表示到模板聚合酶合成物的另一个核苷酸核素，或核苷酸核素的混合体。在一些实施例中，“流循环”指加起来的或者迭代或者并行的四个核苷酸核素，其中例如一个流循环包括所有四个核苷酸核素的总和。

[0056] 当绘制了一个流程图，每个流的检测光或其它信号的值可以是大约零（意味着流中的核苷酸核素不是下一个序列位置处模板中核苷酸核素的补体，因而未被合并），或大约 1（意味着检测到刚好合并一个核苷酸核素补体到模板中的核苷酸核素中），或者大约大于

1 的整数（意味着检测到在模板中并入了两个或多个在流互补的两个连续核苷酸核素中出现的核苷酸核素的副本）。

[0057] 如上所述，重复系列流的理论结果导致从每个流的信号应该正好为零，或整数并且表示在完美流程图中。尽管不同的试验变量包括 CF 和 IE 机制，通过改变数量，真实检测的信号趋向于围绕着这些期望的理论值波动。包括这个变量的被检测信号表示为不清晰的或观察的流程图。

[0058] 这里术语流程图和热解图是可以相互交替使用的。术语“完美流程图”，“干净流程图”和“理论流程图”在这里也可以交替使用。术语“不清晰的流程图”，“现实生活流程图”和“可观察的流程图”在这里也可以交替使用。

[0059] 另外，如这里所用的，“读”一般是指从单个核酸模板分子或多个基本上相同模板分子的副本的群体中获得的整个序列数据。“未成熟的分子”一般是指通过合并作为模板分子中对应核苷酸核素补体的核苷酸核素，正在由依赖于模板的 DNA 聚合酶延伸的 DNA 链。这里使用的术语“完成效率”是指在给定流中被完全延伸的未成熟分子的百分比。这里使用的术语“未完成延伸率”通常是指未能完全延伸的未成熟分子的数目和所有未成熟分子数目的比率。

[0060] 如上所述的发明的一些实施例校正每个流被检测的信号，以考虑如上所述的 CF 和 IE 机制。例如，发明的一方面包括对任何已知序列计算相位同步损失的范围，假设 CF 和 IE 的级别已知。

[0061] 表格 1，如下所示，提供了用数学法模型化的提供 99％或更高准确度的 IE 和 CF 的阈值的例子（如，读至少 99％表示了模板分子的真实序列）用于不同的读取长度。在表 1 中表示的预测值举例说明了对于各种读取长度在排序精确度上的 CF 和 IE 效果的影响和可以忍受的以实现近似 99％的读取精确度的 IE 和 CF 误差的范围。表 1 示出了对于不正确的读取，不大于 1％ CF 率是可以允许的（假设 IE 对于此群体等于零），以为了大约 100 个序列位置达到 99％准确度的读取长度（即，99％或更高的完成效率）。此外，不超过 0.25％的 IE 率是允许的（假设 CF 率等于零）以为了成为 99％准确率的大约 100 个序列位置的读取长度。

[0062] 表格 1. 导致在不同读取长度下 99％准确性的预测的误差率

[0063]

读取长度（基准）	100		200		400	
不完全延伸	0.0	0.0025	0.0	0.0013	0.0	0.0007
结转	0.1	0.0	0.005	0.0	0.003	0.00
预测精度	~99%	~99%	~99%	~99%	~99%	~99%

[0064] 将要理解的是，表格 1 表示的值是为了解释目的，而不应该被认为限制目的。本领域中的那些普通技术人员将会理解，一些因素可能有助于值的变化性如染色体或参考序列和其他的用于说明预测的参数。例如，SBS 方法典型的实施例通常达到 CF 比率的范围从 1-2％，而 IE 比率的范围从 0.1-0.4％（例如，完成效率范围为 99.6-99.9％）。如上所述，校正 CF 和 IE 是合意的，这是因为损失的相位同步对于读取长度有累积效应，并且随着读取长度增加降低了读取的质量。

[0065] 在一个当前描述的发明的实施例中，表示 CF 和 IE 两者的值被假设成在基本上相同模板分子群体的整个读取期间是基本上不变的，该群体例如是驻留在铬尖晶石系统的单个井中的模板分子的群体。这允许在整个读取期间使用两个简单参数“未完成延伸”和“结转”而不用任何模板分子的实际序列的现有技术对每个序列位置进行数字校正。当前描述的本发明实施例的系统和方法在确定和校正出现在模板分子群体中的 CF 和 IE 的数量时是有用的。例如，本发明的实施例为驻留在每个井中的基本相同的模板分子的每个群体，校正从每个流中检测的信号值，以考虑 CF 和 IE

[0066] 本发明的实施例模拟缺少相位同步作为非线性映射。

[0067] 方程 (1)：

[0068] $M(p, \varepsilon, \lambda) = q$

[0069] 其中：

[0070] -M 是 CAFIE 映射

[0071] -p 是假设的“完美”流程图［如阵列］

[0072] $-\lambda$ 是完成效率参数

[0073] $-\varepsilon$ 是结转参数

[0074] -q 是“不清晰的”流程图［如阵列］

[0075] 理论上的“完美”流程图可以通过使用在方程 (1) 中给出的映射模型公式被转化为现实生活的“不清晰的”流程图以估计 IE 和 CF。用于这种映射公式的模型可以如下方式产生，例如，通过测序具有已知序列的多核苷酸模板分子来分析被提供给可观察流程图 (q) 的误差。在方程 (1) 中给出的数学模型的例子可以在图 1 中举例说明。

[0076] 例如在图 1 的左手边，理论上的流程图 101 是理论上的（完美或理想的）流程图 (p) 的举例说明的表示，其中示出在相邻其相关的核苷酸核素的括号中描述的理想化的信号强度值。理论流程 101 的每个理想化的值是整数或零。在当前例子中，“1”的值代表由单个核苷酸并入得出的 100％检测的信号强度，“0”代表 0％信号（例如，在一个井中包含 1 百万个基本上相同模板分子的群体和 1 百万个未成熟分子，“1”代表当每个未成熟分子被单独的核苷酸延伸时得到的信号，“2”代表当每个未成熟分子被两个核苷酸延伸时得到的信号等）。

[0077] 在图 1 的右手边，可观察的流程 103 是一个从观察的（或模拟的不清晰的）流程图 (q) 中检测到的信号强度值的形象表示。同样的，流程 103 中的每个信号强度值在紧挨着它的相关核苷酸核素的括号中描述。同样在图 1 右手边是流 105，其提供表示与核苷酸核素和信号值（如，流 105 的每次迭代代表在洗涤处理后核苷酸核素的增加物）相关的迭代流序列的代表性的数。例如，在图 1 中说明的流 1 与在所述的流 105 的迭代中介绍的“C”核苷酸核素相关，并且对应于理论流 101 和可观察的流 103 两者的信号值。

[0078] 在图 1 例子中，用于每个流 105 迭代的在理论流程图 101 和可观察的流程图 103 之间的信号强度值中的区别至少部分表示相位同步的损失。例如，在可观察的流程图 103 中代表的信号值不是整数，而是每个更典型的稍微更高或稍微更低于在对于流 105 相同迭代的理论流程图 101 中代表的理想值。

[0079] 表示为“M”的映射模型 110 可以用已知参数 113 的值进行估算。例如，参数 113 包括 ε（结转）参数和 λ（完成效率）参数。参数 113 可以被采用以估算映射模型 110，并

将理论流程图 (p)101 的信号值转换成观察值 (q)103。在当前例子中，由映射模型 110 表示的误差值随着流 105 的每次迭代而积累，并且会成指数增长。

[0080] 继续上述例子，由误差值表示的误差可以在理论上随着每个流成指数增长。例如，在流迭代后，跟基本上相同模板分子的每个群体相关的相位同步测序反应会成为三种不同的相位同步子群体。该子群体包括：相位同步反应的第一子群体，其中流中的核苷酸核素能够恰当的并入在跟模板分子相关的适当的序列位置上（例如，没有 CAFIE 效果）；相位同步反应的第二子群体，其中根据 CF 机制不恰当的并入已经发生并且反应是在相对于第一群体的序列位置之前；和相位同步反应的第三子群体，其中根据 IE 机制不恰当的并入已经发生，并且反应是在第一群体的序列位置之后。在当前例子中，在下个流迭代中，三个子子群体将会从如上所述的三个子群体的每个子群体中形成，等等。本领域那些普通技术人员将会理解，在第 n 个流迭代中，就会有 3^n 相位同步反应的群体，每个在流 n 会产生信号。

[0081] 进一步继续上述例子，图 2 提供了反转在图 2 中表示的映射模型 110 的示例性表示作为反转映射模型 210。例如，通过为参数 113 估计校正值（如用于 ε（结转）和 λ（完成效率）参数这两者的值），观察流程图 (q)103 的信号值被反转回来，以提供理论流程图 (p)101 的信号值。

[0082] 本领域相关技术人员将会理解，图 1 和 2 中表示的信号值被提供仅仅用于举例说明的目的，宽范围的信号值也是可能的。因此这不应该被认定为限制。

[0083] 本发明的一些实施例在下面概述的两连续的阶段 (i) 和 (ii) 中执行反转映射：

[0084] 对于每一个核苷酸核素流 i：

[0085] (i)- 通过核苷酸核素的添加延伸未成熟的分子：

[0086] $$\left\{\begin{array}{c} q_i = \lambda \sum_j m_j p_j \\ (m_j, m_{j'}) \leftarrow (m_j, m_{j'}) + \lambda(-1,1)\, m_j p_j \end{array}\right\} \text{ for all } j \text{ such that } N_j = N_i \text{ and } p_j > 0$$

[0087] 对于所有 j 都有 $N_j = N_i$ 并且 $p_j>0$

[0088] (ii)- 通过从前一个添加中剩余的核苷酸核素延伸未成熟的分子：

[0089] $$\left\{\begin{array}{c} q_i \leftarrow q_j + \varepsilon \sum_j m_j p_j \\ (m_j, m_{j'}) \leftarrow (m_j, m_{j'}) + \varepsilon(-1,1)\, m_j p_j \end{array}\right\} \text{ for all } j \text{ such that } N_j = N_{i-1} \text{ and } p_j > 0$$

[0090] 对于所有 j 都有 $N_j = N_{i-1}$ 并且 $p_j>0$

[0091] 其中

[0092] $-p_i$ 是在第 i 个核苷酸核素流处的理论（干净）流信号值

[0093] $-q_i$ 是在第 i 个核苷酸核素流处的观察的（不清晰的）流信号值

[0094] $-m_i$ 是在对于第 i 个核苷酸核素流的流序列位置上可供并入的核苷酸核素分子的一小部分

[0095] $-N_i$ 是第 i 个核苷酸核素添加 (A, C, G 或 T)

[0096] -(j, j') 是指数对，使得在流上 $p_{j'}$ 是 p_j 的下一个绝对值

[0097] 映射模型按照流对流的形式进行这些计算（例如，迭代流 105）和通过阶段 (i) 和 (ii) 递归式地更新观察流程图 (q) 和部分模板分子 m。

[0098] 图 3a 提供为矩阵计算采用的模型的示例性例子。例如下面将会进行更详细的描述，正向矩阵模型 310 可以被采用以获得反转矩阵模型 320。在当前例子中，使用反转矩阵

模型 320 进行矩阵运算可以被采用以得到对参数 113 的估算。譬如，参数 113 的各种值可以被运用到矩阵计算和评估观察流程图 103 的匹配程度。典型的，提供最匹配观察流程图 (q)103 的参数 113 被认为是对参数 113 的实际值的好的估算。

[0099] 另外，图 3b 提供一个用正向矩阵模型 310 的正向矩阵计算的示例性的例子。在当前例子中，可观察的流程图 (q)103 是由用参数 113 的矩阵计算产生的，参数 113 包括完成效率值 λ ＝ 0.95 和结转值 ε ＝ 0.05。与矩阵的流 105 的迭代相关的每行记录了对每个核苷酸核素流的递归阶段 (i，ii) 的操作和结果。

[0100] 方程 (1) 和递归阶段 (i，ii) 可以被重写为矩阵－阵列操作：

[0101] 方程 (2)[M(p’，ε，λ)]*p ＝ q

[0102] 其中：

[0103] -[M(p’，ε，λ)] 是矩阵

[0104] -* 是矩阵阵列乘法

[0105] -p’ ＝ sgn(p)，是理论或“完美”流的二进制编码（例如，图 1 中的流 p，p ＝ [010200103012]^{t} 将被编码为 p′ ＝ [010100101011]^{t}).

[0106] 方程 (2) 的反转形式给出了反转映射，将“不清晰的”观察流 (q)103 转换回到理论流 (p)101：

[0107] 方程 (3)：p ＝ [M^{-1}(p’，ε，λ)]*q

[0108] 其中：

[0109] _[M^{-1}(p’，ε，λ)] 是（设置－理论）反转矩阵

[0110] 使用迭代法来解决反转方程 (3)，图 3a 中图示为反转矩阵模型 320，来获得理论流 (p)101 用于每个读取。对于 CAFIE 反转，用给定的参数对 113(ε，λ) 来执行这种迭代：

[0111] 方程 (4)：

[0112] $p^{(n+1)}$ ＝ [M^{-1}(p’(n)，ε，λ)]*q

[0113] 其中 $p'^{(n)}$ ≡ sgn($p^{(n)}$－阈值）以及 $p^{(1)}$ ≡ q 被用作为计算的依据。该阈值依赖于系统的信噪比率。

[0114] 和图 3b 相似，图 4a 提供了用反转矩阵模型 320 反转矩阵计算的示例性例子。在当前例子中，理论上干净的流程图 (p)101 从观察的不清晰的流程图 (q)103 中产生，其使用了包括完成效率值 lambda ＝ 0.95 和结转值 epsilon ＝ 0.05 的参数 113。

[0115] 例如，在一个执行中，可以使用固定值，即阈值≡ 0.2。在这样一个执行中，当流程图值 P 大于 0.2 的时候，二进制编码流程图 P’编码值“1”，和当流程图值 p 小于或等于 0.2 时编码值“0”。在当前例子中，阈值 0.2 是信噪比率的估计。

[0116] 可替换的，一些执行方法可以采用在区间 0 和 1 的阈值，如 0.05，0.1 或 0.3。因此，对于给定的参数对 113(ε，λ)，“不清晰的”观察的流程图 (q)103 可以通过方程 (4) 被反转回干净的“完美”理论流程图 (p)101。在很多执行中，一般情况下流程图执行的单次迭代就足够了。在一些执行中，执行 2，3 或更多的流程反转迭代是所希望的，其中随着每次迭代，流程图表示的准确性得到提高，尤其是对更长的读取长度，直到以想要的质量在解决方案上计算收敛。在优选的实施例中，可以为了计算效率而执行流程图反转的一次迭代或二次迭代。同样，用计算机代码实现的本发明的一些实施例可以允许使用者选择一定量的迭代来响应于用户选择而执行和／或系列执行每次迭代。例如，用户可以使用本领域中已知

的方法如在 GUI 中显示的一个或多个栏或选择按钮中输入值来执行选择。在当前例子中，使用者可以输入表示迭代数目的值来执行和 / 或使用者可以选择按钮来执行本发明的迭代。另外，使用者可以选择数据质量的指示，其中本发明迭代直到数据质量的级别达到满意为止。

[0117] 图 4b 提供了一个使用方程 4 的方法在连续次的迭代后结果如何可以被提高的示例性的例子。粗流程图 410 示例了一个具有参数值 113 的观察的流程图 (q)103 的实施例，其包括来自核苷酸核素增加物的 336 流迭代的完成效率值 $\lambda = 0.997$ 和结转值 $\varepsilon = 0.03$，每次迭代由流条 409 表示。例如，每个流条 409 是核苷酸核素流的代表，每个核素可以由条 409 的颜色或图案特别表示。另外，和每个流相关的被检测或校正的信号值由与由信号强度 405 给定的比例相关的条 409 的高度表示。

[0118] 本领域中的那些普通技术人员将会理解，在粗流程图 410 中对于流条 409 的信号强度 405 的值有很大程度的多样性，尤其读取长度大于与由读取长度 407 给定的比例相关的 50 序列位置。换言之，大多数的流条 409 的信号值不包括整数的信号值。流程图 420 的两个迭代示例了在用发明实施例校正的两次迭代之后观察的流程图 (q)103 的相同实施例。对流条 409 的信号强度 405 的一致性得以提高，尤其是对于读取长度 407 位置 105 或更少的流条 409。相似的数据质量的提高分别在 4 次迭代流程图 430 和 8 次迭代流程图 440 中展示，其中流程图 440 示例了基本上所有流条 409 显示一致性和整数值。

[0119] 在一些实施例中，对于参数 113 的估算值可以用方程 (4) 确定。例如，完成效率参数（λ）的最合适值可以通过使用方程 (4) 输入用于完成效率参数不同的值同时使用 CF 参数固定值执行测试计算来确定。在当前例子中，值 $\lambda = 1, 0.999, 0.998, \cdots, 0.990$，和固定 CF 值 $\varepsilon = 0$ 可以被连续使用并且每次得到结果。在不同的实施例中，在输入 λ 值之间的 0.001 的间隔可以被其他间隔取代，例如，间隔值 0.05，0.01，0.005，0.001，0.0005，0.0001，等等。

[0120] 继续当前例子，如果对于流条 409 在计算的理论流程图 (p) 中任何信号值 405 在用输入值 λ 解方程 (4) 后落在零以下，那么那个 λ 值将被宣布为最合适的完成效率参数值。一旦决定了最合适值 λ，那么用后续的更小 λ 值将会导致所谓的“过于合适”和产生人工 - 负的流信号。同样在当前例子中，对于在表示均聚物的长系列的流条 409（例如，一系列的序列位置包括同样的核苷酸核素）之后的序列位置的一些流条 409 的被校正信号值 405 可能落在零以下。这个零交叉点可以被展示在图 5 中的椭圆形 503 中，之后最合适完成效率会被表示成 λ^*。

[0121] 同样，在一些实施例中，CF 的效果可以用类似方法得到。例如，CF 参数值可以被测试，譬如，其可以包括值 $\varepsilon = 0, 0.0025, 0.005, 0.0075, 0.01 \ldots, 0.04$ 和在前面得到的值 λ^* 处固定的完成效率参数 λ。这在图 5 中举例说明，如步骤 2 → 3，其中椭圆形 503 指示起始点 $2(\varepsilon, \lambda) = (0, \lambda^*)$。在当前例子中，在输入值 ε 间的 0.0025 间隔表示用于举例说明的目的，并且可被其他更小的间隔值取代，例如，间隔值 0.05，0.01，0.005，0.001，0.0005，0.0001，0.00001，等等。如果在使用输入值 ε 解答方程 (4) 之后（例如，在沿着 λ 路径搜索期间，流条 409 的任何信号值 405 不同于落到零以下的流条 409 的信号值 405），在计算的理论流程图 (p) 中任何流条 409 的信号值 405 落到零以下，然后那个 ε 值被宣布为最佳匹配 CF 参数值。一旦决定最佳匹配值 ε，用后面更大的值将会导致过佳和人工产生负

的流信号。同样在当前例子中，在表示均聚物的长系列流条 409 之前的序列位置处的一些流条 409 的被校正信号值 405 可能落到零以下。这个零交叉点可以在图 5 椭圆形 505 中示出，最佳匹配 CF 可以在以后被表示成 ε^*。

[0122] 图 5 提供了一个示例性范例，例如，横坐标代表完成效率轴 520，纵坐标代表 CF 轴 510。在椭圆形 501，503 和 505 中的图像每个代表上述步骤并包含示出 3 个信号的流的示例性部分。譬如，中央条代表主信号条 537，两侧是左次要信号 (CF 条 535) 和右次要信号 (IE 条 533)。椭圆形 501 示例了原始可观察流程图 (q)103 的步骤，其主信号条 537 被相位异步缩小，CF 条 535 和 IE 条 533 的次要信号代表由相位异步造成的噪音。椭圆形 503 代表当 IE 已经被校正的步骤，其中与 IE 条 533a 相关的信号被消除，中央主信号条 537 相应地增加。如上所述，IE 已经被校正的点例如包括最佳匹配完成效率参数的零交叉点并表示为 λ^*。椭圆形 505 代表了 CF 已经被校正的更进一步的步骤，其通过消除与 CF 条 535a 相关的信号来举例说明，并且中央主信号条 537 相应地增加。如上所述，CF 已经被校正的点可以例如包括最佳匹配完成效率参数的零交叉点并被表示为 ε^*。椭圆形 505 表示校正结果，其近似于理论的、期望的流程图，其中已经基本上移除了归因于相位异步误差的噪音。

[0123] 因而，因为 CF 和 IE 的数量和潜在的模板分子序列 p 事前是未知的，因此本发明的方法可以被使用在完全的再次分析模式。不需要聚合酶并入效率（即，λ）或核苷酸洗涤的有效性（即，ε）的现有知识；也不需要任何需要执行反转的参考核苷酸序列。

[0124] 在一些实施例中，上述的参数估算查找过程通过阶段 (i，ii) 在每个输入查找间隔 ε 和 λ 处构建了矩阵 [M]，其从计算效率的角度进行限制。这种限制可以至少部分地通过使用在矩阵构建操作上的近似值进行克服。例如，可以避免在每个搜索间隔上重新构建矩阵，从而很大程度提高了计算速度。两种方法如下所述：

[0125] 方法 1：

[0126] 在小的 ε 值和 (1-λ) 上（例如，(1-λ)<＝0.001 且 ε<＝0.0025)，矩阵 [M] 被分解，且近似于形式：

[0127] 方程 (5)：

[0128]

$$[M(p',\varepsilon,\lambda)] \sim [L(p',\Delta\lambda)]^{\Phi} * [U(p',\Delta\varepsilon)]^{\varpi},$$

[0129] 其中：

$$\varpi \sim \varepsilon / \Delta\varepsilon$$

[0130] _Δε＝0.0025 且 Δλ＝0.001，是各自 ε 轴和 λ 轴的间隔。

[0131] _Φ 和 ϖ 是矩阵幂，具有 $\varpi \sim \varepsilon/\Delta\varepsilon$ 以及 $\Phi \sim (1-\lambda)/\Delta\lambda$. 的特征。

[0132] _[L(p’，Δλ)] 是较低对角矩阵，其模拟在小缺陷 (deficiency) Δλ 上 IE 的效果。

[0133] _[U(p’，Δλ)] 是较高对角矩阵，其模拟在小缺陷 Δε 上 CF 的效果。

[0134] 通过这种分解，方程式 (5) 仅仅沿着搜索路径构建了较低对角矩阵 L 和较高对角矩阵 U，以及在搜索坐标方格（ε，λ）处，不完全和结转的程度由矩阵（ϖ，Φ）的幂模拟。该搜索间隔中的小值，Δε＝0.0025 以及 Δλ＝0.001，可以由其他小值替换，例如，0.05，0.01，0.005，0.001，0.0005，0.0001，等等。

[0135] 代替之前展示的（ε，λ）- 坐标方格上的搜索，这里本发明通过一组（ϖ，Φ）- 坐标方格进行，该坐标方格是帮助计算矩阵幂的更适宜的正整数。最适宜的匹配（ϖ^*，Φ^*）是

在零交叉条件下定义的；该对应的完成效率和 CF 参数是 $\lambda^* = (1-\phi^*\Delta\lambda)$ 和 $\varepsilon^* = \varpi^*\Delta\varepsilon$.

[0136] 方法 2：

[0137] 下列方程式 (5) 在小 ε 和 (1-λ) 情况中出现，较低和较高对角矩阵 $[L]^{\phi}$ 和 $[U]^{\varpi}$，进一步近似为

[0138] 方程式 (6)：$[L]^{\phi} \equiv ([1]+[1])^{\phi} \sim [I]+\phi[1]$

[0139] 方程式 (7)：$[U]^{\varpi} \equiv ([I]+[u])^{\varpi} \sim [I]+\varpi[u]$

[0140] 其中：

[0141] -[I] 是相同矩阵。

[0142] -[1] 和 [u] 是 [L] 和 [U] 各自的非对角矩阵。

[0143] 这说明了计算矩阵幂的旁路阶段，且因此提供了进一步的加速（例如，减少）计算时间。该（ϖ，ϕ）中的搜索空间现在包含所有正实数。该最匹配的（ϖ^*，ϕ^*）被定义在零交叉条件下；对应的完成效率和 CF 参数是 $\lambda^* = (1-\phi^*\Delta\lambda)$ 和 $\varepsilon^* = \varpi^*\Delta\varepsilon$.

[0144] 这里提供的实施例是基于构建和反转矩阵，和在（ε，λ）平面中的二维搜索以探测 CAFIE 参数的理想对。在基本上相同的模板分子的每个群体上进行这些计算，这些计算例如可以包括在铬尖晶石类系统中的逐井分析。在一些实施例中，为每个群 / 井构建一个矩阵以产生理想的 CAFIE 值（ε^*，λ^*）。图 6 提供了示例性的例子，其中在一些成百上千群体 / 井 603 的样品中分配完成效率参数 605 值 λ^* 和 CF 参数 607 值 ε^*，如通过使用如上描述的反转 / 查找方法 1 计算的。使用上述方法 2 计算，其需要比方法 1 更少的计算时间，可以提供类似结果。

[0145] 上述的实施例同样可以假设在整个测序试验中跟恒定的完成效率 λ 和 CF ε 参数相关的比率保持恒定。此假设可以通过将 CAFIE 查找和反转程序应用到包括几个流循环（这里“几个”是指在 1 和流循环的总数之间的任何整数）的流程图中的所谓的“流窗口”上而减轻。例如，每个流窗口是流程图中表示的全组流循环的子集，具有一对需要被发现的 CAFIE 参数和相对应的干净的理论流程图 101。在当前例子中，每个流窗口被排列以便它从与测序试验相关的流程图中的第一个流开始并且在某个短于或等于流程图中的流循环的全长的流处结束，其中每个更小的流窗口套入在大一点的窗口中。对于每个流窗口 n 来说，查找和反转处理独立发生以产生一组 CAFIE 参数 113，该 CAFIE 参数 113 现在是窗口索引 n 的功能：$\varepsilon^* = \varepsilon^*(n)$ and $\lambda^* = \lambda^*(n)$.。同样被套入的计算的干净理论流程图 101，p(n)，是这些依赖于索引 n 的 CAFIE 参数的变量值的结果。“缝纫”处理：p = p(n) 用于窗口 (n-1) 和 n 间的流，将流窗口序列 p(n) 重新组合成最终干净的流程图 (p)101。

[0146] 在同样的或可替换的实施例中，λ 和 ε 恒定值的假设可以用另一种方法消除。例如，对于每个核苷酸核素增加物“N”（“A”，”G”，“C”，或“T”），完成效率 λ 和 CF ε 参数可以假设成参量的形式，如指数，并且作为流位置的功能“f”(1,2,3, ……)

[0147] $\lambda_N(f) = \lambda^0_N * \exp(-\delta_N * f)$，

[0148] $\varepsilon_N(f) = \varepsilon^0_N * \exp(-\beta_N * f)$.

[0149] 其中：

[0150] -$\lambda_N(f)$ 是第 f 个流上核苷酸核素“N”的完成效率

[0151] -$\varepsilon_N(f)$ 是第 f 个流上核苷酸核素“N”的 CF

[0152] $-\lambda^{0}_{N}$ 和 ε^{0}_{N} 是初始值

[0153] $-\delta_{N}$ 和 β_{N} 是衰减率。

[0154] 搜索方法被应用在 4 个参数空间中，$\lambda_{N}(0)$，$\varepsilon_{N}(0)$，δ_{N} 和 β_{N}，来确定最佳值。

[0155] 另外，本领域那些普通技术人员将会理解其他不与描述的 CAFIE 机制相关的噪音源也可以存在。这些噪音源可以包括但不局限于，电子源如所谓“暗流”、光源、生物源、化学源或其他本领域已知或将来可能发现的源。当前描述的本发明的一些实施例可以展示对其他噪声源改变的敏感程度，在很多应用中，该敏感程度可以是基本上一致的和 / 或可预知的级别。例如，对于归属于已知和未知来源的可预见的和一致级别的噪音通常很容易校正。其中一种校正法是通过数学加和减来自于与流相关的所有信号值中的与噪音相关的值（取决于是否噪音添加超出信号或减小检测的信号）。

[0156] 在一些实施例中，噪音级别是不可预测的，至少部分不可预测，可以从嵌入在信号数据中的信息中获得噪音级别的估算值。例如，对于已知的或预测不在序列位置上的核苷酸核素，期望实际信号值应该等于零。所以，任何检测到的信号应该归属于系统内所有噪音的来源。在当前例子中，因为当前描述的发明估计噪音形成 CAFIE 机制，因此这种噪音可能从数据或显示的潜在噪音中去除。在当前例子中，估算可以通过考虑测序试验中所有的“零 - 最大有效生产率”(zero-mer) 序列位置来改进。在此情况下，在二进制编码 p’ 方程 (4) 中的“阈”值可以被每个试验动态确定，以表示它的噪音级别，而不是在上述之前的实施例中描述的一个固定值。

[0157] 甚至于，本发明的一些实施例可以包括那些所谓的“安全标准”用以防止在观察的流程图中表示的序列数据的过度校正。如上所述，过度校正可能造成随着所述算法迭代引入的误差上的指数累积。例如，上面描述的其他噪音源可以确定包括将被使用在信号数据上的校正量的安全标准。例如，一些执行方法可以假设来自于其他无 CAFIE 源的给定级别的噪音，并且将所谓的 60％校正度（例如，100％意味着完全校正）的安全标准应用给数据。此估算运用“混合”流程图，“0.6p+0.4q”，包含 60％的计算的干净流程图 p 和 40％的观察的不清晰流程图 q。可替换的，如果无 CAFIE 噪音在“低”级别上，则可以应用更高百分比的校正，例如 80％。

[0158] 例 1

[0159] 在 454 生命科学基因组测序器上对金黄色葡萄球菌 COL 和生殖支原体的基因组进行鸟枪法测序 (Margulies 等，2005，结合上述内容作为参考）。图 7 提供了一种在基因组覆盖范围上仅仅 IE 校正和 CAFIE 校正的效果的例证性的例子，校正共有序列、介质读取长度、和具有对超过 125 个序列位置的读取长度实现 100％准确度的井的百分比。通过这些措施中的每一个措施，CAFIE 校正优于单独的 IE 校正。单独的 IE 校正优于在没有校正的情况下实现的结果。在准备阵列之前，具有控制序列的珠子被准备与试验样品分离和混合。

[0160] 通过使用以上流程，63 循环试验的平均读取长度从 112 序列位置增加到 147 序列位置，该 147 序列位置靠近 63 循环或 252 流迭代的理论最大值（例如每个流循环包括 4 个核苷酸核素流迭代）。平均起来，在 4 个核苷酸添加的每个循环中，通过相乘流循环的数目在这种情况下是 63 和被延伸的序列位置的数目 (2.5) 计算理论最大值 :63x2.5 = 157.5（理论最大值）。147 序列位置平均的读取长度通过映射流程图到已知的基因组序列来确定，在流循环上具有 95％的精确度。

[0161] 进一步，在此公开的是，使用上述方法 1 的四个示例性的由如上所述的数据处理应用执行的伪码计算机程序，即：

[0162] (1)buildTransitionMatrixIEOnly.c

[0163] 构造用于不完全延伸的转移矩阵

[0164] (2)buildTransitionMatrixCFOnly.c

[0165] 构造用于结转的转移矩阵

[0166] (3)cafieCorrectOneNukeTraceFastTMC2.c

[0167] 反转在 (1) 中计算的转移矩阵，并且搜索 IE 值

[0168] (4)cafieCorrectOneNukeTraceFastCarryForwardOnly.c

[0169] 反转在 (2) 中计算的转移矩阵，并且搜索 CF 值

[0170] 输入是用于每个读取的不清晰的流程图和流顺序（核苷酸添加）；输出是干净的流程图和最佳值（ε^*, λ^*）。将要理解的是，这些伪码计算机程序仅仅是例证性的，并且各种修改和变化落入本发明的范围内。

[0171] 由此可见提供了方法和系统用于校正在核酸测序期间获得的序列数据中的误差。尽管具体的实施例已经在此详细地公开，但这只是为了例证说明起见举例来说，并不意味着对于随后附加的权利要求范围进行限制。具体来讲，可以进行各种替换，改变和修改而不会脱离正如权利要求所定义的本发明的精神和范围。其它方面、优点、和修改被认为是落入下列权利要求的范围之内。提供的权利要求书表示这里公开的发明。其它未经要求保护的发明也可以被预期到。在此保留在以后的权利要求中追加本发明的权利。

[0172] 计算机程序清单

[0173]

计算机程序清单 1：buildTransitionMatrixCFOnly

```
/*****************************************************************************
**
** Function Name: buildTransitionMatrixCFOnly
**
*****************************************************************************
** Description:
**
**    This function generates a square matrix for a given nuke signal trace
**       and given extension completion efficiency and carry over factor.
**
*****************************************************************************
**
** Inputs:
**                numPositives    -       number of positive flows - dimesion of the matrix
**                eff                             -       extension completion efficiency
**                carryOver               -       carry over factor
**
** Outputs:
**                u                               -       N x N matrix, where N is the
number of nuke flows
**
** Returns:
**                none
**
```

[0174]

```
*******************************************************************************/

/*******************************************************************************
**
** Includes
**
*******************************************************************************/
#include <cafieCommons.h>
/*******************************************************************************
**
** Function Declarations
**
*******************************************************************************/

/*******************************************************************************
**
** Global Declarations
**
*******************************************************************************/

/*******************************************************************************
**
** Entry Point
**
*******************************************************************************/
void buildTransitionMatrixCFOnly (struct listTracking *nukeTraceList,
                                  double carryOver,
                                  int *numPositives,
                                  int **positives,
                                  int *numCallables,
                                  int **callables,
                                  double ***outputMatrix)
{
/*******************************************************************************
**
** Variable declarations
**
*******************************************************************************/
    static char functionName[] = "buildTransitionMatrixCFOnly";
    static int traceLevel      = TRACE_LEVEL_8;

    int listError;
    int i;
    int j;
    int index;
//  int numPositives;
    int numNukeFlows;
    int numSoFar;
    int unCondensedIndex;

    int *index1;
    int index1Num = 0;
    int *index2;
    int index2Num = 0;
    struct wellInfo **index2Flows;
```

[0175]

```
        BOOL sequenceEnded;
        double eff = 1.0;
        double df;
        double maxMarginalValue;
        int maxMarginalIndex;
        int testMarginalIndex;
        struct wellInfo *maxMarginalFlow;
        double *fractionAtEachFlow;
#if USE_DELTA_BUFFER
        double *deltaFractionAtEachFlow;
#endif
        double **fM;
        double **condensedMatrix;

        struct wellInfo *fromNukeList;
        struct wellInfo *fromPosList;
        struct wellInfo *prevNukeInfo = NULL;
/*****************************************************************************
**
** Begin Executable Code
**
*****************************************************************************/
        traceIn (functionName, traceLevel);

        /*
        **      Get the number of nuke flows
        */
        numNukeFlows = listGetCount (nukeTraceList, &listError);

        *positives = index1 = safeMalloc (sizeof(int) * numNukeFlows);
        *callables = index2 = safeMalloc (sizeof(int) * numNukeFlows);
        index2Flows = safeMalloc (sizeof(struct wellInfo *) * numNukeFlows);

        /*
        **      Get the number of positive nuke flows
        */

        /*
        **      Allocate space for a rectangular matrix of dimension:
        **      numNukeFlows x numPositives
        */
        fM = safeMalloc (numNukeFlows * sizeof (double *));
        for (i = 0,sequenceEnded = FALSE;
                i < numNukeFlows;
                i++)
        {
                fM[i] = safeMalloc (numNukeFlows * sizeof (double));
                fromNukeList = listGetAt (nukeTraceList,i,&listError);
                if (fromNukeList->signal > TMC_LOWER_CUTOFF)
                {
#if 1
                        fM[i][i] = 1.0;
#endif
                        index1[index1Num] = i;
                        index1Num++;
                        if (fromNukeList->signal > TMC_UPPER_CUTOFF)
```

[0176]

```
                {
                    index2[index2Num] = i;
                    index2Flows[index2Num] = fromNukeList;
                    index2Num++;
                }
            }
            /*
            ** Check to make sure this wouldn't be three negatives in a row
            ** which will totally hose up the correction.  Since we know we
            ** key pass don't worry about the boundry with no positives
            */
#if FORCE_CALLS
            if ((i - index1[index1Num-1] == 3) && !sequenceEnded)
            {
                /*
                ** Promote the highest value from index1[index1Num-1]+1 to i to a
0.2 positive
                */
                maxMarginalIndex = index1[index1Num-1]+1;
                fromNukeList = listGetAt
        (nukeTraceList,maxMarginalIndex,&listError);
                maxMarginalValue = fromNukeList->signal;
                for (testMarginalIndex = maxMarginalIndex+1;
                        testMarginalIndex <= i;
                        testMarginalIndex++)
                {
                    fromNukeList = listGetAt
        (nukeTraceList,testMarginalIndex,&listError);
                    if (fromNukeList->signal > maxMarginalValue)
                    {
                        maxMarginalValue = fromNukeList->signal;
                        maxMarginalIndex = testMarginalIndex;
                    }
                }
#if 1
                fM[maxMarginalIndex][maxMarginalIndex] = 1.0;
#endif
                index1[index1Num] = maxMarginalIndex;
                index1Num++;
            }
            if ((i - index2[index2Num-1] == 3) && !sequenceEnded)
            {
                /*
                ** Promote the highest value from index1[index1Num-1]+1 to i to a
0.5 singlet
                */
                maxMarginalIndex = index2[index2Num-1]+1;
                fromNukeList = listGetAt
        (nukeTraceList,maxMarginalIndex,&listError);
                maxMarginalValue = fromNukeList->signal;
                maxMarginalFlow = fromNukeList;
                for (testMarginalIndex = maxMarginalIndex+1;
                        testMarginalIndex <= i;
                        testMarginalIndex++)
                {
```

[0177]

```
                                fromNukeList = listGetAt
        (nukeTraceList,testMarginalIndex,&listError);
                                if (fromNukeList->signal > maxMarginalValue)
                                {
                                        maxMarginalValue = fromNukeList->signal;
                                        maxMarginalIndex = testMarginalIndex;
                                        maxMarginalFlow = fromNukeList;
                                }
                        }
                        if (maxMarginalValue > TMC_TOO_LOW_TO_FORCE)
                        {
                                index2[index2Num] = maxMarginalIndex;
                                index2Flows[index2Num] = maxMarginalFlow;
                                index2Num++;
                        }
                        else
                        {
                                sequenceEnded = TRUE;
                        }
                }
#endif
        }

        /*
        ** Return the actual numbers of positives and callables
        */
        *numCallables = index2Num;
        *numPositives = index1Num;

        condensedMatrix = safeMalloc    (sizeof(double *) * index1Num);
        *outputMatrix = condensedMatrix;
        for (i = 0; i < index1Num; i++)
        {
                condensedMatrix[i] = safeMalloc (index1Num * sizeof (double));
        }

#if DUMP_INFO
        for (i = 0; i < index1Num; i++)
        {
                fprintf   (stderr,"%s - index1 pos '%d'\t'%d'\n",functionName,i,index1[i]);
        }
        fprintf   (stderr,"\n");
        for (i = 0; i < index2Num; i++)
        {
                fprintf   (stderr,"%s - index2 pos '%d'\t'%d'\n",functionName,i,index2[i]);
        }
        fprintf   (stderr,"\n");

        fprintf   (stderr,"%s - Whole matrix before calculation\n",functionName);
        for (i = 0; i < numNukeFlows; i++)
        {
                for (j = 0; j < numNukeFlows; j++)
                {
                        fprintf   (stderr,"\t%.4lf",fM[i][j]);
                }
                fprintf   (stderr,"\n");
```

[0178]

```
        }
#endif


#if 0
        /*
        **      Set all matrix elements to zero
        */
        for (i = 0; i < index1Num; i++)
        {
                for (j = 0; j < index1Num; j++)
                {
                        u[i][j] = 0.0;
                }
        }
#endif

        /*
        **      Allocate space to hold incorporation info
        */
        fractionAtEachFlow = safeMalloc (index2Num * sizeof (double));
#if USE_DELTA_BUFFER
        deltaFractionAtEachFlow = safeMalloc (index2Num * sizeof (double));
#endif
        fractionAtEachFlow[0] = 1.0;

        for (i = 0; i < numNukeFlows; i++)
        {
                /*
                **      First pass of positiveTraceList calculates incomplete extention
                */
                fromNukeList = listGetAt (nukeTraceList, i, &listError);
#if USE_DELTA_BUFFER
                memset (deltaFractionAtEachFlow,0,sizeof(double)*index2Num);
#endif
                for (j = 0; j < index2Num; j++)
                {
                        fromPosList = index2Flows[j];
                        unCondensedIndex = index2[j];

                        if (fromNukeList->fluidFlowed == fromPosList->fluidFlowed)
                        {
                                df = fractionAtEachFlow[j] * eff;

#if USE_DELTA_BUFFER
                                deltaFractionAtEachFlow[j] -= df;
#else
                                fractionAtEachFlow[j] -= df;
#endif

                                fM[i][unCondensedIndex] = df;
                                if (j != (index2Num - 1))
                                {
#if USE_DELTA_BUFFER
                                        deltaFractionAtEachFlow[j+1] += df;
#else
```

[0179]

```
                                        fractionAtEachFlow[j+1] += df;
#endif
                                    }
                                }
                            }
#if USE_DELTA_BUFFER
                            for (j = 0; j < index2Num; j++)
                            {
                                fractionAtEachFlow[j] += deltaFractionAtEachFlow[j];
                            }
#endif

                            /*
                            **      Second pass calculates carryforward (i > 0)
                            */
                            if (i > 0)
                            {
#if USE_DELTA_BUFFER
                                memset (deltaFractionAtEachFlow,0,sizeof(double)*index2Num);
#endif
                                for (j = 0; j < index2Num; j++)
                                {
                                    fromPosList = index2Flows[j];
                                    unCondensedIndex = index2[j];

                                    if (prevNukeInfo->fluidFlowed == fromPosList-
>fluidFlowed)
                                    {
                                        df = fractionAtEachFlow[j] * carryOver;
#if USE_DELTA_BUFFER
                                        deltaFractionAtEachFlow[j] -= df;
#else
                                        fractionAtEachFlow[j] -= df;
#endif
                                        fM[i][unCondensedIndex] = df;

                                        if (j != (index2Num - 1))
                                        {
#if USE_DELTA_BUFFER
                                            deltaFractionAtEachFlow[j+1] += df;
#else
                                            fractionAtEachFlow[j+1] += df;
#endif

                                        }
                                    }
                                }
#if USE_DELTA_BUFFER
                                for (j = 0; j < index2Num; j++)
                                {
                                    fractionAtEachFlow[j] += deltaFractionAtEachFlow[j];
                                }
#endif
                            }

                            prevNukeInfo = fromNukeList;
```

[0180]

```
        }

        /*
        **      Truncate to condensed space
        */
#if DUMP_INFO

        fprintf   (stderr,"%s - Whole matrix after calculation\n",functionName);
        for (i = 0; i < numNukeFlows; i++)
        {
                for (j = 0; j < numNukeFlows; j++)
                {
                        fprintf   (stderr,"\t%.4lf",fM[i][j]);
                }
                fprintf   (stderr,"\n");
        }

        fprintf   (stderr,"%s - EFF-'%lf', CF-'%lf'\n",functionName,eff,carryOver);
        for (i = 0; i < index1Num; i++)
        {
                fromPosList = listGetAt   (nukeTraceList,index1[i],&listError);
                fprintf   (stderr,"\t%.4lf",fromPosList->signal);
        }
        fprintf (stderr,"\n");
#endif
        index = 0;
        for (i = 0; i < index1Num; i++)
        {
                for (j = 0; j < index1Num; j++)
                {
                        condensedMatrix[i][j] = fM[index1[i]][index1[j]];
#if DUMP_INFO
                        fprintf   (stderr,"\t%.4lf",condensedMatrix[i][j]);
#endif
                }
#if DUMP_INFO
                fprintf   (stderr,"\n");
#endif
        }


        safeFree (fractionAtEachFlow);
#if USE_DELTA_BUFFER
        safeFree (deltaFractionAtEachFlow);
#endif
        safeFree (index2Flows);

        if (fM)
        {
                for (i = 0; i < numNukeFlows; i++)
                {
                        safeFree (fM[i]);
                }
                safeFree (fM);
        }



                traceOut (functionName, traceLevel);
                return;
        }
```

[0181] 计算机程序清单 2 :buildTransitionMatrixIEOnly

[0182]

```
/*****************************************************************************
**
** Function Name: buildTransitionMatrixIEOnly

**
*****************************************************************************
** Description:
**
**   This function generates a square matrix for a given nuke signal trace
**     and given extension completion efficiency and carry over factor.
**
*****************************************************************************
**
** Inputs:
**              numPositives    -       number of positive flows - dimesion of the matrix
**              eff                             -       extension completion efficiency
**              carryOver               -       carry over factor
**
** Outputs:
**              u                               -       N x N matrix, where N is the
number of nuke flows
**
** Returns:
**              none
**
*****************************************************************************/

/*****************************************************************************
**
** Includes
**
*****************************************************************************/
#include <cafieCommons.h>
/*****************************************************************************
**
** Function Declarations
**
*****************************************************************************/

/*****************************************************************************
**
** Global Declarations
**
*****************************************************************************/

/*****************************************************************************
**
```

[0183]

```
** Entry Point
**
*****************************************************************************/
void buildTransitionMatrix1EOnly (struct listTracking *nukeTraceList,
                                  double eff,
                                  int *numPositives,
                                  int **positives,
                                  int *numCallables,
                                  int **callables,
                                  double ***outputMatrix)
{
/*****************************************************************************
**
** Variable declarations
**
*****************************************************************************/
        static char functionName[] = "buildTransitionMatrix1EOnly";
        static int traceLevel     = TRACE_LEVEL_8;

        int listError;
        int i;
        int j;
        int index;
//      int numPositives;
        int numNukeFlows;
        int numSoFar;
        int unCondensedIndex;

        double maxMarginalValue;
        int maxMarginalIndex;
        int testMarginalIndex;
        struct wellInfo *maxMarginalFlow;
        int *index1;
        int index1Num = 0;
        struct wellInfo **index1Flows;
        int *index2;
        int index2Num = 0;
        struct wellInfo **index2Flows;

        double df;
        double *fractionAtEachFlow;
        BOOL sequenceEnded;
#if USE_DELTA_BUFFER
        double *deltaFractionAtEachFlow;
#endif
        double **fM;
        double **condensedMatrix;

        struct wellInfo *fromNukeList;
        struct wellInfo *fromPosList;
        struct wellInfo *prevNukeInfo;
/*****************************************************************************
**
** Begin Executable Code
**
*****************************************************************************/
```

[0184]

```
traceIn (functionName, traceLevel);

/*
**      Get the number of nuke flows
*/
numNukeFlows = listGetCount (nukeTraceList, &listError);

*positives = index1 = safeMalloc (sizeof(int) * numNukeFlows);
*callables = index2 = safeMalloc (sizeof(int) * numNukeFlows);
index2Flows = safeMalloc (sizeof(struct wellInfo *) * numNukeFlows);
index1Flows = safeMalloc (sizeof(struct wellInfo *) * numNukeFlows);

/*
**      Get the number of positive nuke flows
*/

/*
**      Allocate space for a rectangular matrix of dimension:
**      numNukeFlows x numPositives
*/
fM = safeMalloc (numNukeFlows * sizeof (double *));
for (i = 0,numSoFar = 0,sequenceEnded = FALSE;
        i < numNukeFlows;
        i++)
{
        fM[i] = safeMalloc (numNukeFlows * sizeof (double));
        fromNukeList = listGetAt (nukeTraceList,i,&listError);
        if (fromNukeList->signal > TMC_LOWER_CUTOFF && !sequenceEnded)
        {
#if 0
                fM[i][i] = 1.0;
#endif
                index1[index1Num] = i;
                index1Flows[index1Num] = fromNukeList;
                index1Num++;
                if (fromNukeList->signal > TMC_UPPER_CUTOFF)
                {
                        index2[index2Num] = i;
                        index2Flows[index2Num] = fromNukeList;
                        index2Num++;
                }
        }
#if FORCE_CALLS
        /*
        ** Check to make sure this wouldn't be three negatives in a row
        ** which will totally hose up the correction.  Since we know we
        ** key pass don't worry about the boundry with no positives
        */
        if ((i - index1[index1Num-1] == 3) && !sequenceEnded)
        {
                /*
                ** Promote the highest value from index1[index1Num-1]+1 to i to a
0.2 singlet
                */
                maxMarginalIndex = index1[index1Num-1]+1;
```

[0185]

```
                        fromNukeList = listGetAt
        (nukeTraceList,maxMarginalIndex,&listError);
                        maxMarginalValue = fromNukeList->signal;
                        maxMarginalFlow = fromNukeList;
                        for (testMarginalIndex = maxMarginalIndex+1;
                                testMarginalIndex <= i;
                                testMarginalIndex++)
                        {
                                fromNukeList = listGetAt
        (nukeTraceList,testMarginalIndex,&listError);
                                if (fromNukeList->signal > maxMarginalValue)
                                {
                                        maxMarginalValue = fromNukeList->signal;
                                        maxMarginalFlow = fromNukeList;
                                        maxMarginalIndex = testMarginalIndex;
                                }
                        }
#if 0
                        fM[maxMarginalIndex][maxMarginalIndex] = 1.0;
#endif
                        index1[index1Num] = maxMarginalIndex;
                        index1Flows[index1Num] = maxMarginalFlow;
                        index1Num++;
                }
                if ((i - index2[index2Num-1] == 3) && !sequenceEnded)
                {
                        /*
                        ** Promote the highest value from index1[index1Num-1]+1 to i to a
0.2 singlet
                        */
                        maxMarginalIndex = index2[index2Num-1]+1;
                        fromNukeList = listGetAt
        (nukeTraceList,maxMarginalIndex,&listError);
                        maxMarginalValue = fromNukeList->signal;
                        maxMarginalFlow = fromNukeList;
                        for (testMarginalIndex = maxMarginalIndex+1;
                                testMarginalIndex <= i;
                                testMarginalIndex++)
                        {
                                fromNukeList = listGetAt
        (nukeTraceList,testMarginalIndex,&listError);
                                if (fromNukeList->signal > maxMarginalValue)
                                {
                                        maxMarginalValue = fromNukeList->signal;
                                        maxMarginalIndex = testMarginalIndex;
                                        maxMarginalFlow = fromNukeList;
                                }
                        }
                        if (maxMarginalValue > TMC_TOO_LOW_TO_FORCE)
                        {
                                index2[index2Num] = maxMarginalIndex;
                                index2Flows[index2Num] = maxMarginalFlow;
                                index2Num++;
                        }
                        else
                        {
```

[0186]

```
                                sequenceEnded = TRUE;
                            }
                        }
#endif
        }

        /*
        ** Return the number of callables and positives
        */
        *numCallables = index2Num;
        *numPositives = index1Num;

        condensedMatrix = safeMalloc    (sizeof(double *) * index1Num);
        *outputMatrix = condensedMatrix;
        for (i = 0; i < index1Num; i++)
        {
                condensedMatrix[i] = safeMalloc (index1Num * sizeof (double));
        }

#if DUMP_INFO
        for (i = 0; i < index1Num; i++)
        {
                fprintf   (stderr,"%s - index1 pos '%d'\t'%d'\n",functionName,i,index1[i]);
        }
        fprintf   (stderr,"\n");
        for (i = 0; i < index2Num; i++)
        {
                fprintf   (stderr,"%s - index2 pos '%d'\t'%d'\n",functionName,i,index2[i]);
        }
        fprintf   (stderr,"\n");

        fprintf   (stderr,"%s - Whole matrix before calculation\n",functionName);
        for (i = 0; i < numNukeFlows; i++)
        {
                for (j = 0; j < numNukeFlows; j++)
                {
                        fprintf   (stderr,"\t%.4lf",fM[i][j]);
                }
                fprintf   (stderr,"\n");
        }
#endif


#if 0
        /*
        **      Set all matrix elements to zero
        */
        for (i = 0; i < index1Num; i++)
        {
                for (j = 0; j < index1Num; j++)
                {
                        u[i][j] = 0.0;
                }
        }
#endif
```

[0187]

```
        /*
        **      Allocate space to hold incorporation info
        */
        fractionAtEachFlow = safeMalloc (indexINum * sizeof (double));
#if USE_DELTA_BUFFER
        deltaFractionAtEachFlow = safeMalloc (indexINum * sizeof (double));
#endif
        fractionAtEachFlow[0] = 1.0;

        for (i = 0; i < numNukeFlows; i++)
        {
                /*
                **      First pass of positiveTraceList calculates incomplete extention
                */
                fromNukeList = listGetAt (nukeTraceList, i, &listError);

#if USE_DELTA_BUFFER
                memset  (deltaFractionAtEachFlow,0,sizeof(double)*indexINum);
#endif
                for (j = 0; j < indexINum; j++)
                {
                        fromPosList = indexIFlows[j];
                        unCondensedIndex = indexI[j];

                        if (fromNukeList->fluidFlowed == fromPosList->fluidFlowed)
                        {
                                df = fractionAtEachFlow[j] * eff;
#if USE_DELTA_BUFFER
                                deltaFractionAtEachFlow[j] -= df;
#else
                                fractionAtEachFlow[j] -= df;
#endif
                                fM[i][unCondensedIndex] = df;
                                if (j != (indexINum - 1))
                                {
#if USE_DELTA_BUFFER
                                        deltaFractionAtEachFlow[j+1] += df;
#else
                                        fractionAtEachFlow[j+1] += df;
#endif
                                }
                        }
                }
#if USE_DELTA_BUFFER
                for (j = 0; j < indexINum; j++)
                {
                        fractionAtEachFlow[j] += deltaFractionAtEachFlow[j];
                }
#endif
        }

        /*
        **      Truncate to condensed space
        */
#if DUMP_INFO
```

[0188]

```
        fprintf   (stderr,"%s - Whole matrix after calculation\n",functionName);
        for (i = 0; i < numNukeFlows; i++)
        {
                for (j = 0; j < numNukeFlows; j++)
                {
                        fprintf   (stderr,"\t%lf",fM[i][j]);
                }
                fprintf   (stderr,"\n");
        }

        fprintf   (stderr,"%s - EFF-'%lf', CF-'%lf'\n",functionName,eff,0.0);
        for (i = 0; i < index1Num; i++)
        {
                fromPosList = listGetAt   (nukeTraceList,index1[i],&listError);
                fprintf   (stderr,"\t%.4lf",fromPosList->signal);
        }
        fprintf (stderr,"\n");
#endif
        index = 0;
        for (i = 0; i < index1Num; i++)
        {
                for (j = 0; j < index1Num; j++)
                {
                        condensedMatrix[i][j] = fM[index1[i]][index1[j]];
#if DUMP_INFO
                        fprintf   (stderr,"\t%.4lf",condensedMatrix[i][j]);
#endif
                }
#if DUMP_INFO
                fprintf   (stderr,"\n");
#endif
        }


        safeFree (fractionAtEachFlow);
#if USE_DELTA_BUFFER
        safeFree (deltaFractionAtEachFlow);
#endif
        safeFree (index1Flows);
        safeFree (index2Flows);

        if (fM)
        {
                for (i = 0; i < numNukeFlows; i++)
                {
                        safeFree (fM[i]);
                }
                safeFree (fM);
        }

        traceOut (functionName, traceLevel);
        return;
}
```

[0189] 计算机程序清单 3：

[0190]

cafieCorrectOneNukeTraceFastCarryForwardOnly

[0191]

```
/*****************************************************************************
**
** Function Name: cafieCorrectOneNukeTraceFastTMC2
**
**

**
*****************************************************************************
** Description:
**
**    This function reads in a nuke pyrogram information list and perform
**       CAFIE Correction to it.
**
*****************************************************************************
**
** Inputs:
**                 none
**
** Outputs:
**    error           - error value
**
** Input/Outputs:
**                 nukeTraceList  -  binary well trace list
**
** Returns:
**                 none
**
*****************************************************************************/

/*****************************************************************************
**
** Includes
**
*****************************************************************************/
#include <cafieCommons.h>

#define TEST 0
/*****************************************************************************
**
** Function Declarations
**
*****************************************************************************/

/*****************************************************************************
**
** Global Declarations
**
*****************************************************************************/
extern FILE *errorOut;
/*****************************************************************************
**
** Entry Point
```

[0192]

```
**
*****************************************************************************/
double cafieCorrectOneNukeTraceFastTMC2 (struct listTracking *nukeTraceList,
                                         int *error)
{
/*****************************************************************************
**
** Variable declarations
**
*****************************************************************************/
        static char functionName[] = "cafieCorrectOneNukeTraceFastTMC2";
        static int traceLevel      = TRACE_LEVEL_8;

        int i;
        int j;
        int listCnt;
        int index;
        int listError;
        int numFlows;
        int numCallables;
        int numPositives = 0;
        int *indx = NULL;

        double effMax = 0.999;
        double effMin = 0.990;
        double newVal;
        double bestEff;
        double eff;
        double efficiencyUsed = 1.0;
        double sum;
        double carryOver;
        double lambda;
        double maxD;
        double maxQ;
        double numPositiveD;
        double d;

        double *nukeSignals = NULL;
        double *superDirtySignals = NULL;
        double *newSuperDirtySignals = NULL;
        double *tempDubP;
        double *x = NULL;
        double *b = NULL;
        double **condensedMatrix = NULL;
        double **nonLuCondensedMatrix = NULL;

        int *positives = NULL;
        int *callables = NULL;

        struct wellInfo *wellInfo;

/*****************************************************************************
**
** Begin Executable Code
**
*****************************************************************************/
```

[0193]

```
traceIn (functionName, traceLevel);

/*
**      Assume we will be ok
*/
*error = ANALYSIS_OK;

/*
**      Build the transition matrix. Set the carryOver parameter to zero
**      and use the effMax
*/
buildTransitionMatrixIEOnly (nukeTraceList,
                effMax,
                &numPositives,
                &positives,
                &numCallables,
                &callables,
                &condensedMatrix);
numFlows = listGetCount (nukeTraceList, &listError);

/*
**      If there're not enough number (1/4 of total flows) of positives,
**      we won't do anything
*/
if (numCallables <= 0.25 * numFlows)
{
        goto bye;
}

/*
**      Populate it and hold a copy in an array
*/
nukeSignals = safeMalloc (numPositives * sizeof (double));
for (i = 0; i < numPositives; i++)
{
        wellInfo = listGetAt (nukeTraceList,positives[i], &listError);
        if (listError != LIST_OK)
        {
                fprintf (errorOut,
                                "%s - Error getting %d-th wellInfo from
nukeList.\n",
                                functionName,
                                i);
                *error = listError;
                goto bye;
        }
        nukeSignals[i] = wellInfo->signal;
}

/*
**      Allocate space for a copy of nuke signals
*/
b = safeMalloc (numPositives * sizeof (double));
superDirtySignals = safeMalloc (numPositives * sizeof (double));
newSuperDirtySignals = safeMalloc (numPositives * sizeof (double));
for (i = 0; i < numPositives; i++)
```

[0194]

```
        {
                superDirtySignals[i] = b[i] = nukeSignals[i];
        }

        /*
        **      Allocate space for these arrays to be used for solving linear
        **      equations
        */
        indx = safeMalloc (numPositives * sizeof (int));
        x = safeMalloc (numPositives * sizeof (double));
        nonLuCondensedMatrix = safeMalloc (numPositives * sizeof (double *));
        for (i = 0; i < numPositives; i++)
        {
                nonLuCondensedMatrix[i] = safeMalloc (numPositives * sizeof (dou
                for (j = 0; j < numPositives; j++)
                {
                        nonLuCondensedMatrix[i][j] = condensedMatrix[i][j];
                }
        }

        /*
        **      We will solve a linear system of form: AX=B, where A will be u,
        **      X will be the solution vector, and B is initially nukeSignals,
        **      afterwards, it will store the solution vector.
        */
#if DO_LU_DECOMPOSITION

        if (!luDecomposition (condensedMatrix, numPositives, indx, &d))
        {
                *error = ERR_CAFIE_NON_INVERTABLE_MATRIX;
                goto bye;
        }
#endif

        /*
        **      Main trunk of the Condensed Matrix Algorithm below is a loop
        **      process to search for the best extension efficiency parameter
        */
        bestEff = 0.0;
        for (eff = effMax; eff >= effMin; eff -= 0.001)
        {
                /*
                ** Apply the matrix transform
                */
                luBackSubstitution (condensedMatrix, numPositives, indx, b);

                /*
                ** Also create the new, dirtier pyrogram
                */
                for (i = 0; i < numPositives; i++)
                {
                        newVal = 0.0;
                        for (j = 0; j < numPositives; j++)
                        {
                                newVal += nonLuCondensedMatrix[i][j] *
superDirtySignals[j];
```

[0195]

```
        }
        newSuperDirtySignals[i] = newVal;
}

/*
** Swap the pointers
*/
tempDubP = newSuperDirtySignals;
newSuperDirtySignals = superDirtySignals;
superDirtySignals = tempDubP;

/*
** See if we have any negative values. This is the indicator that we
** have corrected enough.
*/
numPositiveD = 0;
for (i = 0; i < numPositives; i++)
{
        if (b[i] < 0.0)
        {
                numPositiveD++;
                break;
        }
}

/*
** If we have a positive count then it is time to jump into the actual
** correction phase
*/
if (numPositiveD > 0)
{
        maxD = fabs (b[0] - nukeSignals[0]);
        for (i = 1; i < numPositives; i++)
        {
                if (maxD < fabs (b[i] - nukeSignals[i]))
                {
                        maxD = fabs (b[i] - nukeSignals[i]);
                }
        }

        maxQ = fabs (superDirtySignals[0] - nukeSignals[0]);
        for (i = 1; i < numPositives; i++)
        {
                newVal = fabs (superDirtySignals[i] - nukeSignals[i]);
                if (maxQ < newVal)
                {
                        maxQ = newVal;
                }
        }

        lambda = 0.8 * maxQ / maxD;
        if (lambda > 1.0)
        {
                lambda = 1.0;
        }
```

[0196]

```
                        /*
                        **      Construct the new nuke signals from solution
                        */
                        for (i = 0; i < numPositives; i++)
                        {
                                x[i] = nukeSignals[i] + lambda * (b[i] - nukeSignals[i]);

                                /*
                                **      Corrected signals
                                */
                                nukeSignals[i] = x[i];
                        }

                        /*
                        **      We found the best efficiency, won't try more stages
                        */
                        efficiencyUsed = bestEff = eff;
                        break;
                }
        }

        /*
        **      If we didn't find the best efficiency, we won't alter nuke trace
        */
        if (bestEff == 0.0)
        {
                goto bye;
        }

        /*
        **      Now make the change official in the original nuke trace
        */
        for (i = 0; i < numPositives; i++)
        {
                wellInfo = listGetAt (nukeTraceList, positives[i], &listError);
                if (listError != LIST_OK)
                {
                        fprintf (errorOut,
                                        "%s - Error getting %d-th wellInfo from
positiveTraceList.\n",
                                        functionName,
                                        i);
                        *error = listError;
                        goto bye;
                }
#if DO_CORRECTION
                wellInfo->signal = nukeSignals[i];
                if (wellInfo->signal < 0.0)
                {
                        wellInfo->signal = 0.0;
                }
#endif
        }

bye:
        /*
```

[0197]

```
**      Free up space
*/
if (positives)
{
        safeFree (positives);
}
if (callables)
{
        safeFree (callables);
}
if (nukeSignals)
{
        safeFree (nukeSignals);
}
if (superDirtySignals)
{
        safeFree (superDirtySignals);
}
if (newSuperDirtySignals)
{
        safeFree (newSuperDirtySignals);
}
if (indx)
{
        safeFree (indx);
}
if (b)
{
        safeFree (b);
}
if (x)
{
        safeFree (x);
}

for (i = 0; i < numPositives; i++)
{
        if (condensedMatrix && condensedMatrix[i])
        {
                safeFree (condensedMatrix[i]);
        }
        if (nonLuCondensedMatrix && nonLuCondensedMatrix[i])
        {
                safeFree (nonLuCondensedMatrix[i]);
        }
}

if (condensedMatrix)
{
        safeFree (condensedMatrix);
}
if (nonLuCondensedMatrix)
{
        safeFree (nonLuCondensedMatrix);
}



        traceOut (functionName, traceLevel);
        return (efficiencyUsed);
}
```

[0198] 计算机程序清单 4 :cafieCorrectOneNukeTraceFastTMC2

[0199]

```
/*****************************************************************************
**
** Function Name: cafieCorrectOneNukeTraceFastTMC2
**
**

**
*****************************************************************************
** Description:
**
**    This function reads in a nuke pyrogram information list and perform
**      CAFIE Correction to it.
**
*****************************************************************************
**
** Inputs:
**                none
**
** Outputs:
**    error          - error value
**
** Input/Outputs:
**                nukeTraceList  -  binary well trace list
**
** Returns:
**                none
**
*****************************************************************************/

/*****************************************************************************
**
** Includes
**
*****************************************************************************/
#include <cafieCommons.h>

#define TEST 0
/*****************************************************************************
**
** Function Declarations
**
*****************************************************************************/

/*****************************************************************************
**
** Global Declarations
**
*****************************************************************************/
```

[0200]

```
extern FILE *errorOut;
/*****************************************************************************
**
** Entry Point
**
*****************************************************************************/
double cafieCorrectOneNukeTraceFastTMC2 (struct listTracking *nukeTraceList,
                                         int *error)
{
/*****************************************************************************
**
** Variable declarations
**
*****************************************************************************/
        static char functionName[] = "cafieCorrectOneNukeTraceFastTMC2";
        static int traceLevel      = TRACE_LEVEL_8;

        int i;
        int j;
        int listCnt;
        int index;
        int listError;
        int numFlows;
        int numCallables;
        int numPositives = 0;
        int *indx = NULL;

        double effMax = 0.999;
        double effMin = 0.990;
        double newVal;
        double bestEff;
        double eff;
        double efficiencyUsed = 1.0;
        double sum;
        double carryOver;
        double lambda;
        double maxD;
        double maxQ;
        double numPositiveD;
        double d;

        double *nukeSignals = NULL;
        double *superDirtySignals = NULL;
        double *newSuperDirtySignals = NULL;
        double *tempDubP;
        double *x = NULL;
        double *b = NULL;
        double **condensedMatrix = NULL;
        double **nonLuCondensedMatrix = NULL;

        int *positives = NULL;
        int *callables = NULL;

        struct wellInfo *wellInfo;

/*****************************************************************************
```

[0201]

```
**
** Begin Executable Code
**
*****************************************************************************/
        traceIn (functionName, traceLevel);

        /*
        **      Assume we will be ok
        */
        *error = ANALYSIS_OK;

        /*
        **      Build the transition matrix. Set the carryOver parameter to zero
        **      and use the effMax
        */
        buildTransitionMatrixIEOnly (nukeTraceList,
                        effMax,
                        &numPositives,
                        &positives,
                        &numCallables,
                        &callables,
                        &condensedMatrix);
        numFlows = listGetCount (nukeTraceList, &listError);

        /*
        **      If there're not enough number (1/4 of total flows) of positives,
        **      we won't do anything
        */
        if (numCallables <= 0.25 * numFlows)
        {
                goto bye;
        }

        /*
        **      Populate it and hold a copy in an array
        */
        nukeSignals = safeMalloc (numPositives * sizeof (double));
        for (i = 0; i < numPositives; i++)
        {
                wellInfo = listGetAt (nukeTraceList,positives[i], &listError);
                if (listError != LIST_OK)
                {
                        fprintf (errorOut,
                                        "%s - Error getting %d-th wellInfo from
nukeList.\n",
                                        functionName,
                                        i);
                        *error = listError;
                        goto bye;
                }
                nukeSignals[i] = wellInfo->signal;
        }

        /*
        **      Allocate space for a copy of nuke signals
        */
```

[0202]

```
        b = safeMalloc (numPositives * sizeof (double));
        superDirtySignals = safeMalloc (numPositives * sizeof (double));
        newSuperDirtySignals = safeMalloc (numPositives * sizeof (double));
        for (i = 0; i < numPositives; i++)
        {
                superDirtySignals[i] = b[i] = nukeSignals[i];
        }

        /*
        **      Allocate space for these arrays to be used for solving linear
        **      equations
        */
        indx = safeMalloc (numPositives * sizeof (int));
        x = safeMalloc (numPositives * sizeof (double));
        nonLuCondensedMatrix = safeMalloc (numPositives * sizeof (double *));
        for (i = 0; i < numPositives; i++)
        {
                nonLuCondensedMatrix[i] = safeMalloc (numPositives * sizeof (double));
                for (j = 0; j < numPositives; j++)
                {
                        nonLuCondensedMatrix[i][j] = condensedMatrix[i][j];
                }
        }

        /*
        **      We will solve a linear system of form: AX=B, where A will be u,
        **      X will be the solution vector, and B is initially nukeSignals,
        **      afterwards, it will store the solution vector.
        */
#if DO_LU_DECOMPOSITION

        if (!luDecomposition (condensedMatrix, numPositives, indx, &d))
        {
                *error = ERR_CAFIE_NON_INVERTABLE_MATRIX;
                goto bye;
        }
#endif

        /*
        **      Main trunk of the Condensed Matrix Algorithm below is a loop
        **      process to search for the best extension efficiency parameter
        */
        bestEff = 0.0;
        for (eff = effMax; eff >= effMin; eff -= 0.001)
        {
                /*
                ** Apply the matrix transform
                */
                luBackSubstitution (condensedMatrix, numPositives, indx, b);

                /*
                ** Also create the new, dirtier pyrogram
                */
                for (i = 0; i < numPositives; i++)
                {
                        newVal = 0.0;
```

[0203]

```
                for (j = 0; j < numPositives; j++)
                {
                        newVal += nonLuCondensedMatrix[i][j] *
superDirtySignals[j];
                }
                newSuperDirtySignals[i] = newVal;
        }

        /*
        ** Swap the pointers
        */
        tempDubP = newSuperDirtySignals;
        newSuperDirtySignals = superDirtySignals;
        superDirtySignals = tempDubP;

        /*
        ** See if we have any negative values.  This is the indicator that we
        ** have corrected enough.
        */
        numPositiveD = 0;
        for (i = 0; i < numPositives; i++)
        {
                if (b[i] < 0.0)
                {
                        numPositiveD++;
                        break;
                }
        }

        /*
        ** If we have a positive count then it is time to jump into the actual
        ** correction phase
        */
        if (numPositiveD > 0)
        {
                maxD = fabs (b[0] - nukeSignals[0]);
                for (i = 1; i < numPositives; i++)
                {
                        if (maxD < fabs (b[i] - nukeSignals[i]))
                        {
                                maxD = fabs (b[i] - nukeSignals[i]);
                        }
                }

                maxQ = fabs (superDirtySignals[0] - nukeSignals[0]);
                for (i = 1; i < numPositives; i++)
                {
                        newVal = fabs (superDirtySignals[i] - nukeSignals[i]);
                        if (maxQ < newVal)
                        {
                                maxQ = newVal;
                        }
                }

                lambda = 0.8 * maxQ / maxD;
                if (lambda > 1.0)
```

[0204]

```
                        {
                                lambda = 1.0;
                        }

                        /*
                        **      Construct the new nuke signals from solution
                        */
                        for (i = 0; i < numPositives; i++)
                        {
                                x[i] = nukeSignals[i] + lambda * (b[i] - nukeSignals[i]);

                                /*
                                **      Corrected signals
                                */
                                nukeSignals[i] = x[i];
                        }

                        /*
                        **      We found the best efficiency, won't try more stages
                        */
                        efficiencyUsed = bestEff = eff;
                        break;
                }
        }

        /*
        **      If we didn't find the best efficiency, we won't alter nuke trace
        */
        if (bestEff == 0.0)
        {
                goto bye;
        }

        /*
        **      Now make the change official in the original nuke trace
        */
        for (i = 0; i < numPositives; i++)
        {
                wellInfo = listGetAt (nukeTraceList, positives[i], &listError);
                if (listError != LIST_OK)
                {
                        fprintf (errorOut,
                                        "%s - Error getting %d-th wellInfo from
positiveTraceList.\n",
                                        functionName,
                                        i);
                        *error = listError;
                        goto bye;
                }
#if DO_CORRECTION
                wellInfo->signal = nukeSignals[i];
                if (wellInfo->signal < 0.0)
                {
                        wellInfo->signal = 0.0;
                }
#endif
```

[0205]

```
        }

bye:
        /*
        **      Free up space
        */
        if (positives)
        {
                safeFree (positives);
        }
        if (callables)
        {
                safeFree (callables);
        }
        if (nukeSignals)
        {
                safeFree (nukeSignals);
        }
        if (superDirtySignals)
        {
                safeFree (superDirtySignals);
        }
        if (newSuperDirtySignals)
        {
                safeFree (newSuperDirtySignals);
        }
        if (indx)
        {
                safeFree (indx);
        }
        if (b)
        {
                safeFree (b);
        }
        if (x)
        {
                safeFree (x);
        }

        for (i = 0; i < numPositives; i++)
        {
                if (condensedMatrix && condensedMatrix[i])
                {
                        safeFree (condensedMatrix[i]);
                }
                if (nonLuCondensedMatrix && nonLuCondensedMatrix[i])
                {
                        safeFree (nonLuCondensedMatrix[i]);
                }
        }

        if (condensedMatrix)
        {
                safeFree (condensedMatrix);
        }
        if (nonLuCondensedMatrix)
```

[0206]

```
                {
                        safeFree (nonLuCondensedMatrix);
                }

                traceOut (functionName, traceLevel);
                return (efficiencyUsed);
        }
```

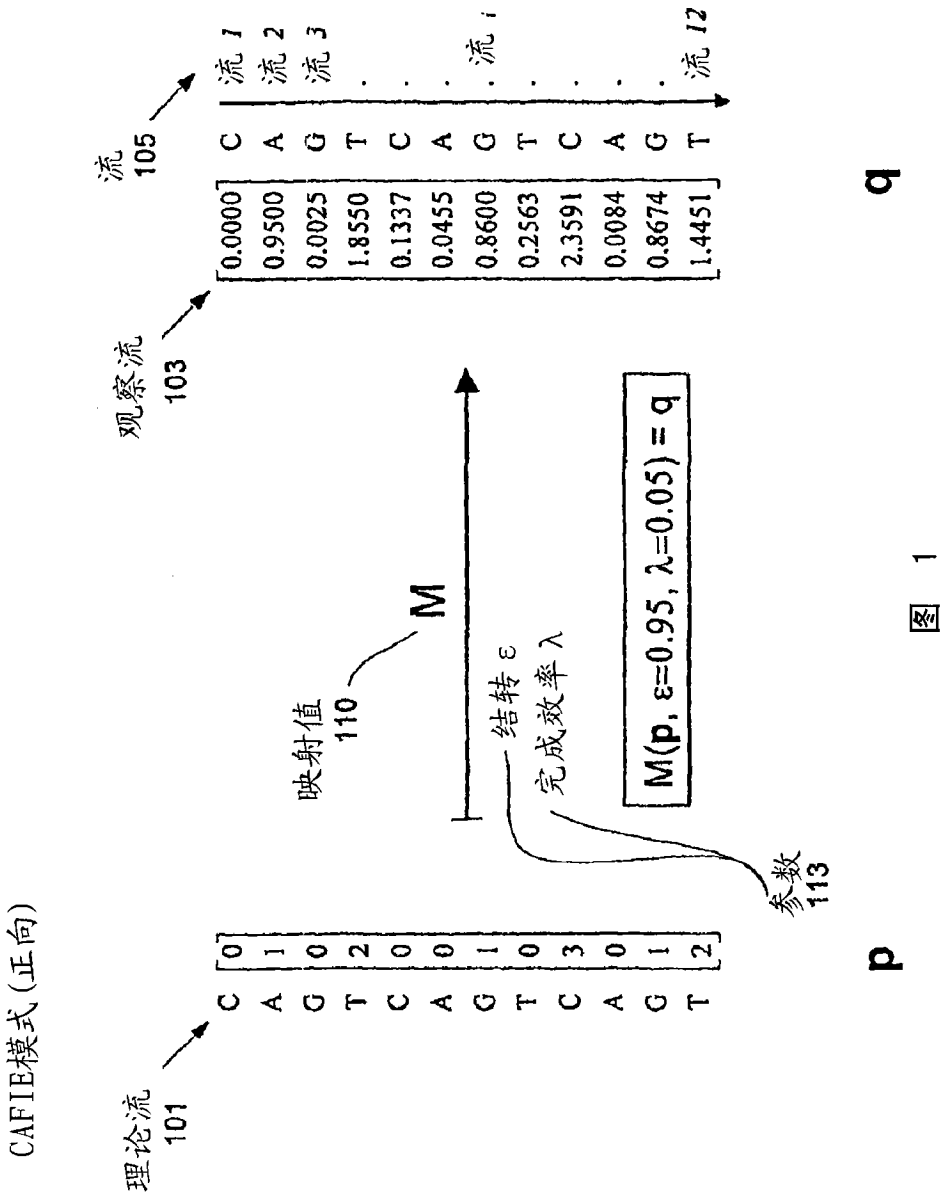

CAFIE模式(正向)
理论流
101
观察流
103
流
105
映射值
110
M
结转 ε
完成效率 λ
参数
113
M(p, ε=0.95, λ=0.05) = q
流 1
流 2
流 3
流 i
流 12
p
q

图 1

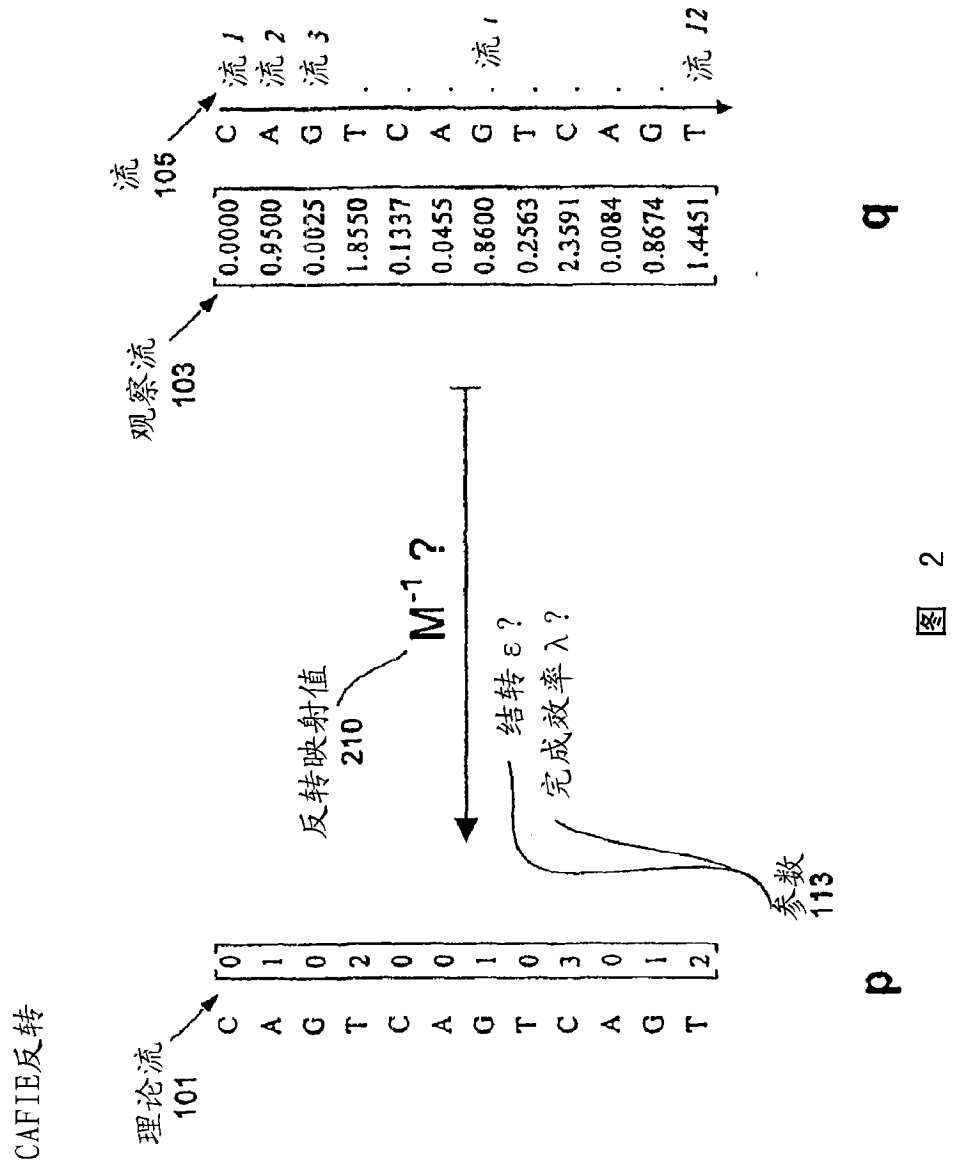

CAFIE反转
理论流
101
观察流
103
流
105
反转映射值
210
M^{-1} ?
结转 ε ?
完成效率 λ ?
参数
113
p
q
流 1
流 2
流 3
流 12

图 2

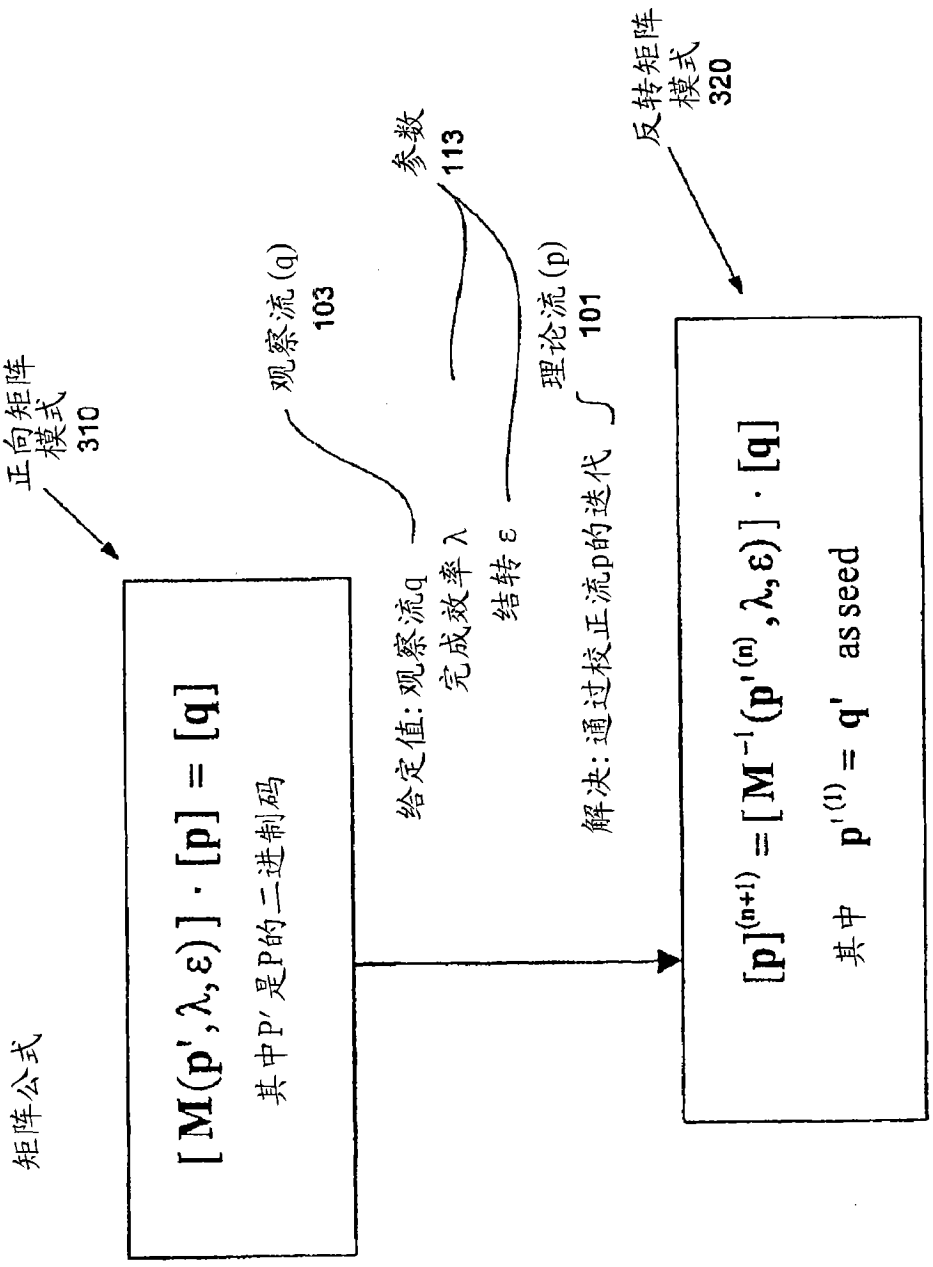

矩阵公式
正向矩阵模式 310
[M(p',λ,ε)] · [p] = [q]
其中P′是P的二进制码
观察流(q) 103
参数 113
给定值:观察流q
完成效率 λ
结转 ε
理论流(p) 101
解决:通过校正流p的迭代
反转矩阵模式 320
[p]^(n+1) = [M^-1(p'^(n),λ,ε)] · [q]
其中 p'^(1) = q' as seed

图 3A

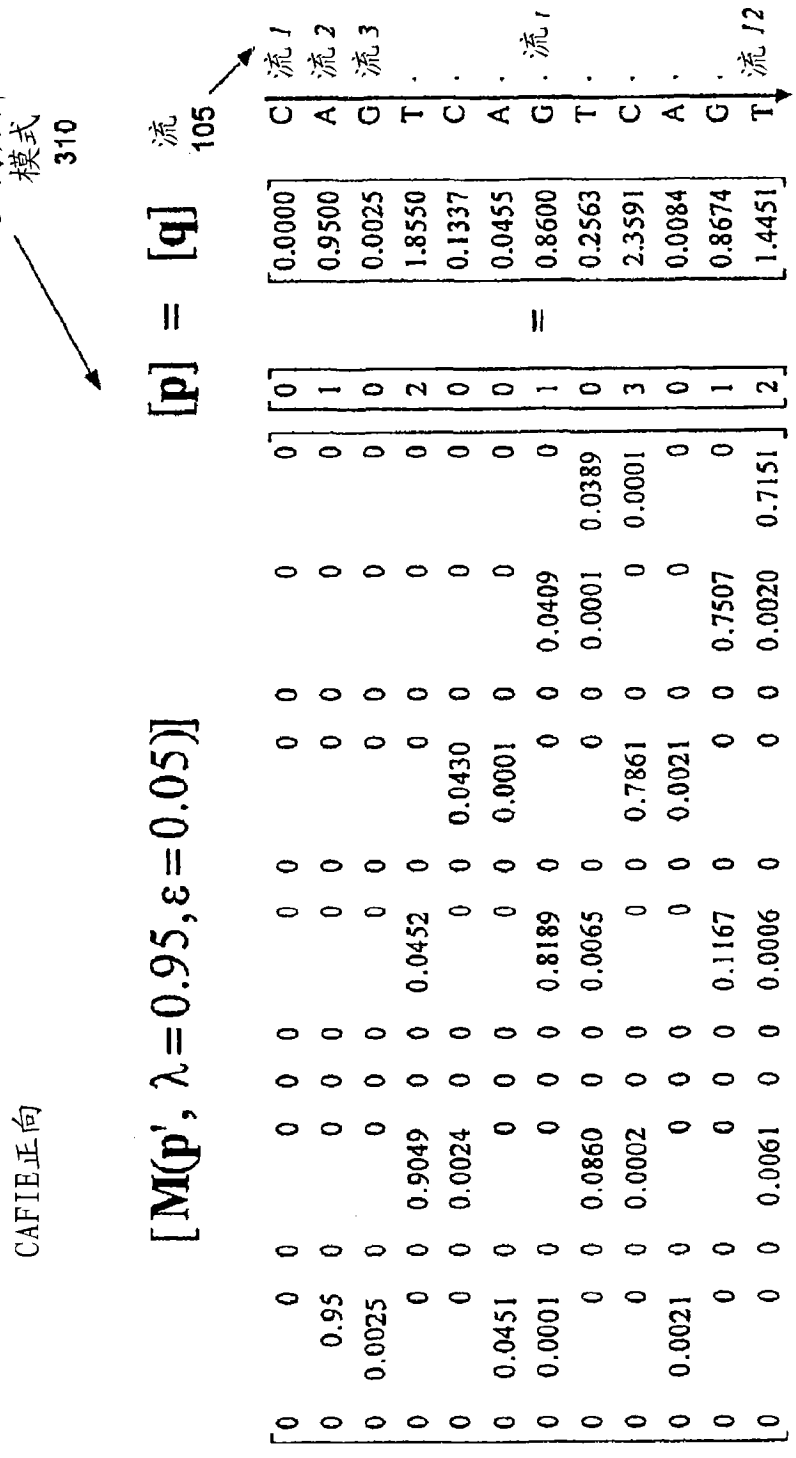


图 3B

图 4A

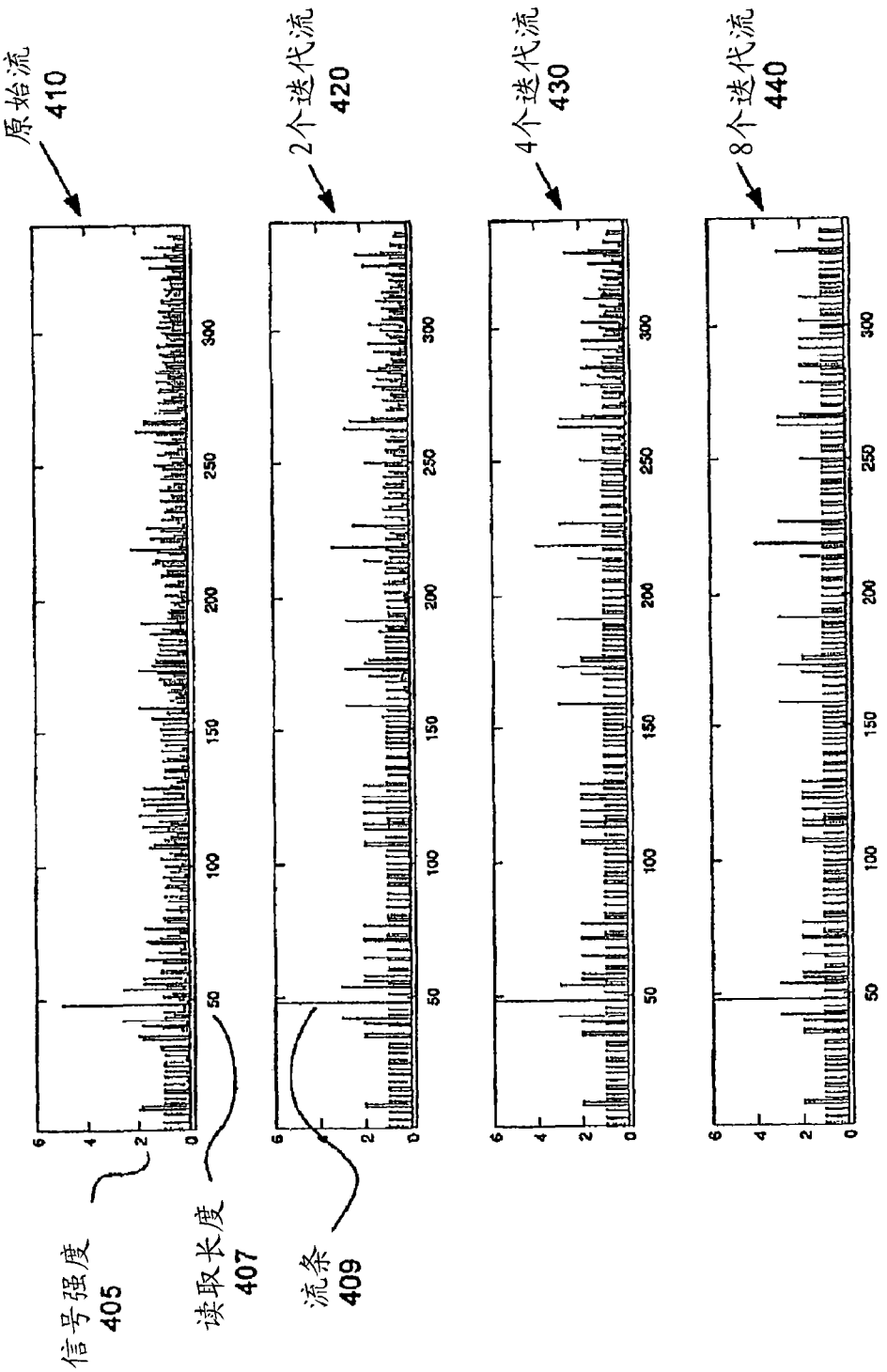


图 4B

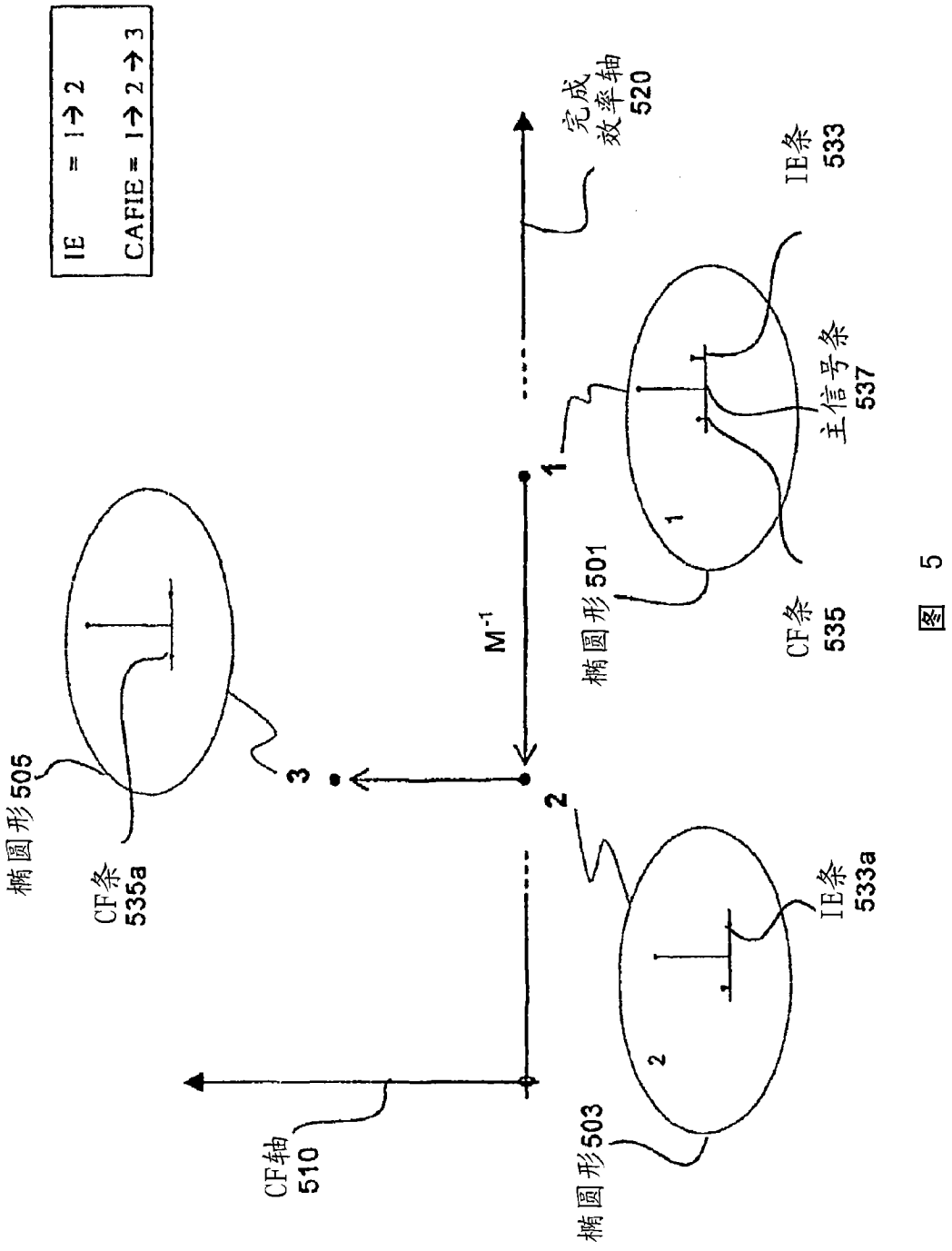

IE = 1→2
CAFIE = 1→2→3
完成效率轴 520
IE条 533
主信号条 537
椭圆形 501
CF条 535
椭圆形 505
CF条 535a
M-1
CF轴 510
椭圆形 503
IE条 533a
1
2
3

图 5

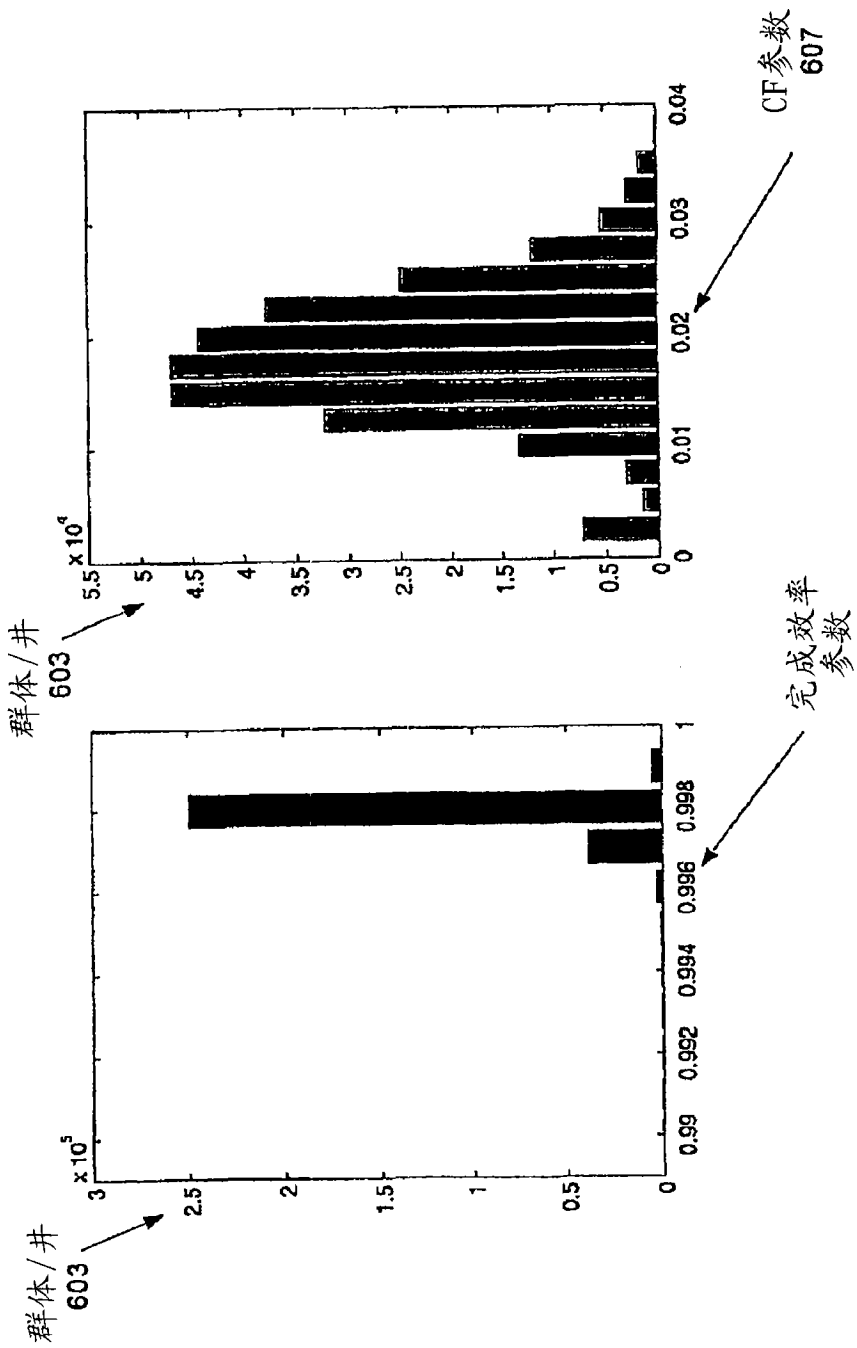

群体/井
603
x10^4
5.5
5
4.5
4
3.5
3
2.5
2
1.5
1
0.5
0
0
0.01
0.02
0.03
0.04
CF参数
607
群体/井
603
x10^5
3
2.5
2
1.5
1
0.5
0
0.99
0.992
0.994
0.996
0.998
1
完成效率
参数
605

图 6

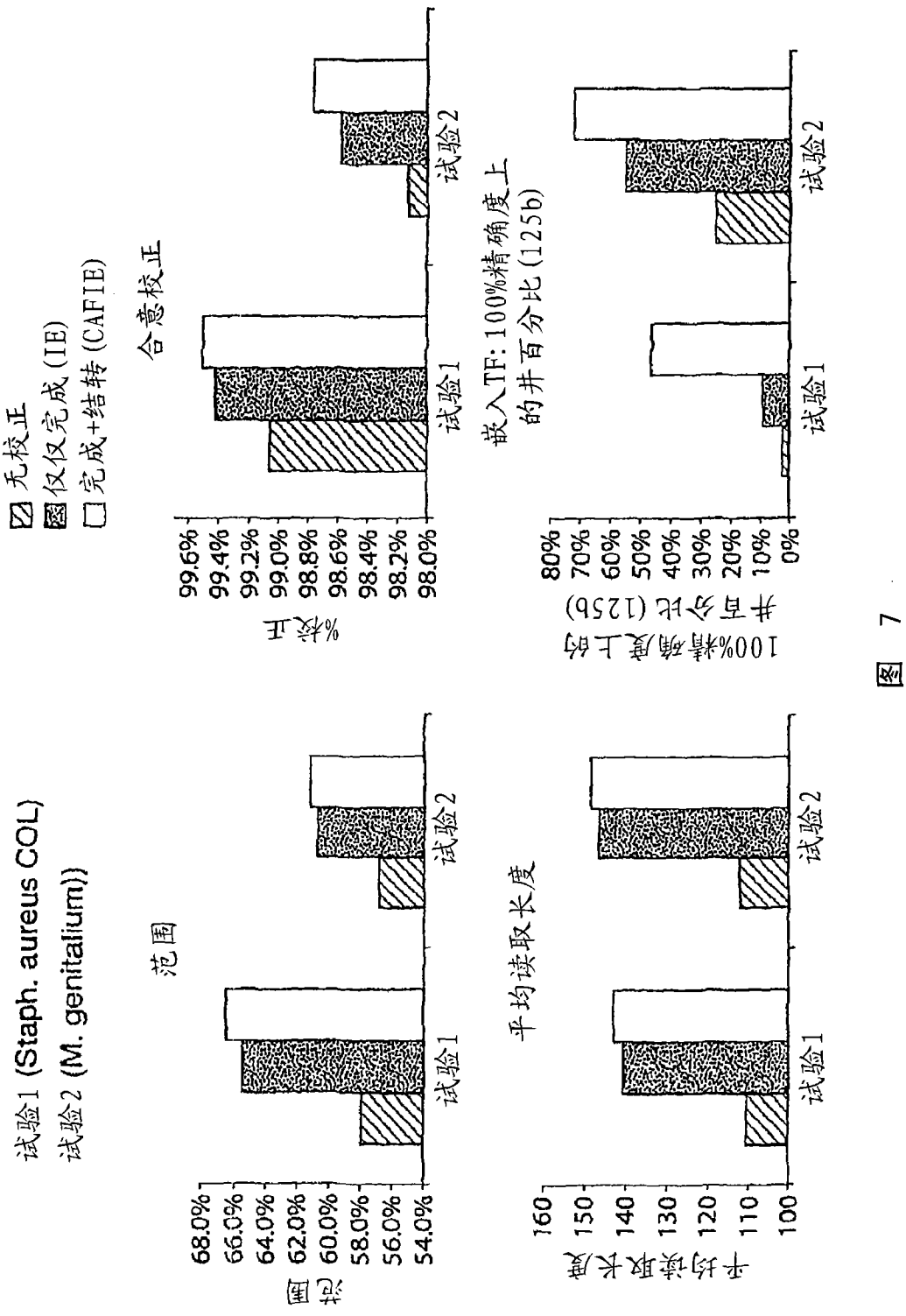

无校正
仅仅完成 (IE)
完成+结转 (CAFIE)
合意校正
%校正
99.6%
99.4%
99.2%
99.0%
98.8%
98.6%
98.4%
98.2%
98.0%
试验1
试验2
嵌入TF: 100%精确度上
的井百分比 (125b)
100%精确度上的
井百分比 (125b)
80%
70%
60%
50%
40%
30%
20%
10%
0%
试验1
试验2
试验1 (Staph. aureus COL)
试验2 (M. genitalium))
范围
范围
68.0%
66.0%
64.0%
62.0%
60.0%
58.0%
56.0%
54.0%
试验1
试验2
平均读取长度
平均读取长度
160
150
140
130
120
110
100
试验1
试验2

图 7