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

(19) World Intellectual Property Organization
International Bureau

PCT

(43) International Publication Date
25 June 2009 (25.06.2009)

(10) International Publication Number
WO 2009/077880 A2

(51) International Patent Classification:
G06F 19/00 (2006.01)

(21) International Application Number:
PCT/IB2008/053565

(22) International Filing Date:
3 September 2008 (03.09.2008)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
61/006,041 17 December 2007 (17.12.2007) US

(71) Applicants *(for all designated States except US):* **BEN GURION UNIVERSITY OF THE NEGEV, RESEARCH AND DEVELOPMENT AUTHORITY** [IL/IL]; P.O. Box 653, 84105 Beer Sheva (IL). **ORBOTECH LTD.** [IL/IL]; P.O. Box 215, 81102 Yavne (IL).

(72) Inventors; and

(75) Inventors/Applicants *(for US only):* **SHIMONY, Eyal Shlomo** [IL/IL]; 17 Siphen Street, 85338 Lehavim (IL). **BRAFMAN, Ronen** [IL/IL]; 132 GivatHaKalaniot, 76832 Ramot Meir (IL). **BEREND, Daniel** [IL/IL]; 27 Mishoul Eckron Street, 84807 Beer Sheva (IL). **COHEN, Shimon** [IL/IL]; 11 Sahlav Street, 74202 Ness Ziona (IL).

(74) Agent: **DR. MARK FRIEDMAN LTD.;** Moshe Aviv Tower, 54th floor, 7 Jabotinsky Street, 52520 Ramat Gan (IL).

(81) Designated States *(unless otherwise indicated, for every kind of national protection available):* AE, AG, AL, AM, **AO, AT, AU, AZ, BA, BB, BG, BH, BR, BW, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DO, DZ, EC,** EE, **EG,** ES, **FI, GB, GD, GE, GH, GM, GT, HN, HR, HU, ID, IL, IN, IS, JP, KE, KG, KM, KN, KP, KR, KZ, LA, LC, LK, LR, LS, LT, LU, LY, MA, MD, ME, MG, MK, MN, MW, MX, MY, MZ, NA, NG, NI, NO, NZ, OM, PG, PH, PL, PT, RO, RS, RU, SC, SD,** SE, **SG, SK, SL, SM,** ST, **SV, SY, TJ, TM, TN, TR,** TT, **TZ, UA, UG, US, UZ, VC, VN, ZA, ZM, ZW**

(84) Designated States *(unless otherwise indicated, for every kind of regional protection available):* ARIPO (BW, GH, GM, KE, LS, MW, MZ, NA, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European (AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HR, HU, IE, IS, IT, LT, LU, LV, MC, MT, NL, NO, PL, PT, RO, SE, SI, SK, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

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

(54) **Title:** OPTIMAL TEST ORDERING IN CASCADE ARCHITECTURES

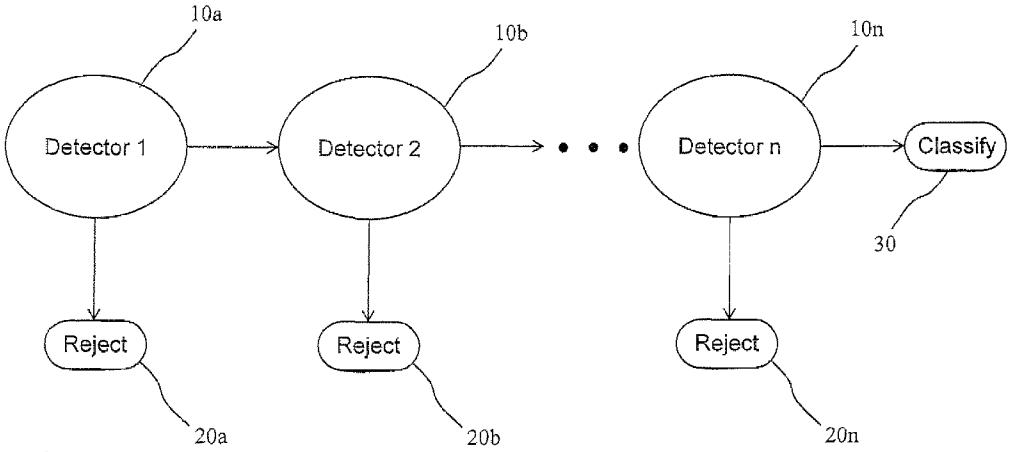

Fig. 1

(57) **Abstract:** Methods for optimizing the cost of executing a set of tests including finding the optimal ordering of the tests for some important cases such as set of tests having series- parallel structure with no statistical dependencies, and near-optimal orderings for the rest of the cases, such that the resources required for executing the tests are minimized.

WO 2009/077880 A2

TITLE OF THE INVENTION

OPTIMAL TEST ORDERING IN CASCADE ARCHITECTURES

RELATED APPLICATION

[0001] This application claims priority to Applicant's U.S. Provisional Patent Appl. No. 61/006,041 titled "FEATURE ORDERING FOR RAPID OBJECT DETECTION" filed Dec 17, 2007, the entirety of which is hereby incorporated by reference herein.

FIELD OF THE INVENTION

[0002] Aspects of the present invention relate to optimal test ordering in cascade architectures, providing a provable method for optimal ordering of tests for some important cases, and near-optimal orderings for the rest of the cases.

BACKGROUND AND PRIOR ART

[0003] In numerous classes of applications, including object detection and acceptance test applications, an important issue is to classify objects or make other decisions in real¬ time. For instance, security systems need to detect targets in real-time and to act on them. Robots need to quickly decide whether an observed artifact is an obstacle, in order to avoid collision. Factory inspection lines must decide as quickly as possible whether or not a manufactured object is faulty. This type of issue has been addressed in work in Machine Learning on the induction of cost-sensitive classifiers, and in particular cost-sensitive decision trees. In this line of research, the goal is to induce a classifier in which the expected cost of tests, and possibly misclassification costs, is minimized. See for example: Marlon N'tfnez. *The use of background knowledge in decision tree induction,* Machine Learning, 6: pages 231-250, 1991; Peter D. Turney, *Cost-sensitive classification: Empirical evaluation of a hybrid genetic decision tree induction algorithm,* Journal of Artificial Intelligence Research (JAIRj, 2: pages 369-409, 1995; and Charles X. Ling, Qiang Yang, Jianning Wang, and Shichao Zhang, *Decision trees with minimal costs,* in International Conference on Machine Learning (ICML), 2004. Most work in this area explores various heuristics and techniques for generating such trees, although Valentina Bayer Zubek and Thomas G. Dietterich, *Pruning improves heuristic search for cost-sensitive learning,* in ICML ,pages

19-26, 2002, is an exception in which an optimal tree results from solving an appropriate Markov Decision Process (MDP). However, the size of the MDP is exponential in the number of attributes, and in general an optimal solution cannot be found.

[0004] In order to simplify the processing, the idea of a cascade system was proposed by Viola: Paul A. Viola and Michael J. Jones, *Rapid object detection using a boosted cascade of simple features,* in Computer Vision and Pattern Recognition (CVPR) (1), pages 511-518, 2001. An example cascade is illustrated in Figure 1 (Prior art). A cascade system is composed of simple detectors 10, each computing one test. For each detector 10, a rejection threshold used for rejection 20 of non-object examples (also called "samples") is learned offline. Difficult examples that pass through the entire sequence of filters arrive at the final stage where an Ada Boost classifier is used to classify those examples into objects or non-objects.

[0005] An important idea behind cascade architectures, as suggested by Viola is to introduce weak classifiers 30 that can classify (hereinafter referred to as "reject") many examples quickly, thus saving considerable computation time, leaving the rest of the examples to be classified later on in the cascade.

[0006] Assuming that a rejection event is always correct, and that all detectors are in the cascade, the classification accuracy is independent of the ordering. Basically, In Viola's scheme, the detectors are ordered such that detectors with high reject probability are placed first, ignoring their runtime. When some detectors require a much larger runtime than others, this becomes problematic in that the resulting runtime is far from optimal.

[0007] There is therefore a need and it would be advantageous to have methods to optimize the runtime, preferably without impinging on overall classification accuracy.

[0008] Related art is described in Eric Horvitz and Jed Lengyel, *Perception, attention, and resources: A decision-theoretic approach to graphics rendering,* in Proceedings of UAI (Conference on Uncertainty in Artificial Intelligence), pages 238-249, August 1997. Methods exemplified by Horvitz et al consist of schemes for reasoning in order to get optimal expected reward, one special case being optimization of expected runtime. But in these methods considering stoppage of a test sequence when a reject is detected is not relevant and has thus not been considered.

[0009] There is therefore a need and it would be advantageous to have methods to optimize tests in a cascade that can detect "rejects" quickly and optimize the runtime of the

[0010] In a cascade, some weak classifiers used in related art, compute features or classifiers as an intermediate computation, creating a structural dependency, which also entails an ordering constraint. Hence it is the intention of the present invention to consider both statistical dependencies and ordering constraints. It is a further intention of the present invention to provide a provably optimal ordering of tests for some important cases, and near-optimal orderings for the rest of the cases.

[0011] The term "ordering constraint," as used with a cascade of tests, refers herein to a prerequisite constraint that a particular test in the cascade run before another particular test. The representation for the ordering constraints is as a partial order. A partial order can also be represented as a directed graph as a notational variant. Whenever *A* must appear before *E* this constraint is denoted by A -> E or by "A before E". Formally, an immediate successor of a test C is a test D, such that there exists no test *Z* with C -> Z -> D. In this case we also refer to *C* as an immediate prerequisite of *D*.

[0012] The term "statistical dependency," as used with a cascade of tests, refers herein to the fact that the reject probability of a test may depend statistically on the results of previously run tests. In a set of tests $X = \{x_1, x_2, \ldots, x_n\}$ conditions under which statistical dependencies between the tests can be handled, are analyzed. Denote $r_{i|S}$ as the probability that test X_i rejects given previous occurrences *S*, where typically *S* would be the reject and/or non-reject of previous tests. For example, $r_{i|y}$ denotes the probability that test x_i does not reject given that X_j has rejected.

DEFINITION OF THE PROBLEM

[0013] Given is a set of tests (alternately called "detectors") $X = \{x_1, x_2, \ldots, x_n\}$ for detecting a certain property, such as a defect in an object, such as a product or a service or a portion thereof. Each test results in a "pass" "reject" or "don't know". In cascade architecture, the object is tested by a sequence of tests (typically a permutation of all the above tests). Also given is an execution time for each test, and a probability of "reject" for each test (possibly conditional on results of previous tests, in the case of statistical dependency). The problem is to find an ordering of the tests in the cascade which is optimal with respect to the total expected runtime of the cascade. It should be noted that time is used as a non-limiting exemplary resource needed to perform tests in the cascade.

[0014] It is possible thai in the cascade computations are re-used by detectors, resulting in ordering constraints between tests. Also, it is possible that tests be statistically dependent. The present invention considers ordering problems under various settings of dependency assumptions of both types.

[0015] The marginal probability that detector x, rejects (necessitating no further processing) is denoted by $r_{,}$, t, denotes the execution time (exemplifying the resources needed to execute test X_{J}) for test x,, and $q_{\iota} = \frac{r_{\cdot}}{t_{\cdot}}$ denotes the "quality" of the test. The reject rate of a sequence S, composed of all tests in any order, is given by:

$$R(s) = 1 - \prod_{i=1}^{n} (1 - r_{\iota}) \tag{1}$$

The expected runtime of the sequence, assuming that the sequence is ordered according to the initial indexing of the tests, is given by:

$$\mathbf{rO)} = \sum_{t=1}^{?!} t_{t} \prod_{J=I}^{\dot{E}-\ I} (1 - 17) \tag{2}$$

(using the convention that a product over an empty set is 1).

The "quality" measure for the sequence as a whole is defined as:

$$Q(S) = \frac{R(S)}{T(S)} . \tag{3}$$

Equation (3) defines the quality of any sequence of tests, consisting of any arbitrary subsequence of X.

[0016] In some variations of the present invention, we allow for a case where a reject decision may be in error. In this case we assume that a test x, falsely rejects examples with a known probability f_{i}. Here the ordering of tests may affect F, the expected number of incorrectly rejected examples. A tradeoff factor C, specified by the user of the system, is assumed. The meaning of this tradeoff is that the user is willing to accept an increased incorrect rejection probability (decreased accuracy) as long as the expected runtime is reduced by at least C time units per unit of decreased accuracy. The goal of the present invention in this extended case is to minimize the expectation of T+C*F.

[0017] It should be noted that the present invention uses the notion of $\frac{r}{t}$ which corresponds to the idea of ratio of incremental gain to computational cost, referred to as

literature, it is not clear in what way our rejection probability can behave like incremental gain. Additionally, in our setting of independent detector ordering, soiling according to $\frac{r}{t}$ gives provably optimal expected time. In the setting of Horvitz, only approximate optimality can be shown. In all other schemes above, even approximate optimality in runtime is not guaranteed. Other prior art, such as US patent 7,050,922 given to Zhengrong Zhou, uses the product *(\-r)*t* and sorts the tests such that the test with smallest *(\-r)*t* is run first. However, such a method results in an expected time that can be far from optimal.

SUMMARY OF THE INVENTION

[0018] Aspects of the present invention includes providing methods of optimizing the cost of executing a set of processes such as detecting defects, or other testing processes, hereinafter collectively and alternately referred to as "tests" or "detectors". The cost of executing each test of the set of tests are the resources invested in running the tests, such as the test execution time, hereinafter collectively and alternately referred to as "runtime" of one or more tests of the set of tests. The set of test is collectively and alternately also referred to as a "cascade" of tests. It should be noted that the runtime is used as a non-limiting exemplary resource needed to perform the tests in a cascade. Each test is capable of detecting or identifying a feature, which if detected, terminates the cascade. Such an occurrence is referred to as a "reject".

[0019] Optimizing the cost of executing a set of tests includes finding the optimal ordering of the tests for some important cases, and near-optimal orderings for the rest of the cases, such that the resources required for executing the tests are minimized.

[0020] In accordance with aspects of the first embodiment of the present invention, a method is provided for an optimal ordering of tests, whereas there are no pre defined order constraints between tests and no statistical dependencies between tests.

[0021] Th accordance with aspects of the second and third embodiments of the present invention, methods are provided for an optimal ordering of tests, whereas there are pre defined order dependencies between tests, but no statistical dependencies between tests.

[0022] In accordance with aspects of the fourth and fifth embodiments of the present invention, methods are provided for an optimal ordering of tests, whereas there are statistical dependencies between tests.

[0023] Additional advantages arid noveTfeatures relating to the present invention will be

set forth In part in the description that follows, and in part will become more apparent to those skilled in the art upon examination of the following or upon learning by practice of aspects of the invention.

BRIEF DESCRIPTION OF THE DRAWINGS

[0024] The present invention will become fully understood from the detailed description given herein below and the accompanying drawings, which are given by way of illustration and example only and thus not limitative of aspects of the present invention discussed here, and wherein:

FIG. 1 (prior art) shows examples cascade of detectors and a classifier;

FIG. 2 schematically illustrates a method for optimal expected runtime of a set of tests in a cascade, in accordance with aspects of embodiments of the present invention, wherein the set of tests includes independent tests with no statistical and no ordering constraint;

FIG. 3 illustrates a graph representation of the following example of tests with ordering constraints: A -> {M, N} -> B;

FIG. 4 illustrates a graph representation of the following example of tests with ordering constraints: a -→ {b -→ c, x -→ y} -→ z;

FIG. 5 illustrates a graph representation of the following example of tests with ordering constraints: d_1 -> {a, b, c} -→ d_2 -→ {x, y, z} -> d_3;

FIG. 6 schematically illustrates a method for optimal expected runtime of a set of tests in a cascade, in accordance with aspects of embodiments of the present invention, wherein the set of tests includes tests in a series-parallel ordering constraint structure with no statistical dependencies;

FIG. 7 illustrates the Conglomeration process used by the method shown in Figure 6.

FIG. 8 illustrates of the following example of ordering constraints: A -> M -→ B and A -> N -→ B and M -→ N;

FIG. 9a illustrates a graph representation of a series-parallel structure of tests X, wherein X = {A, B, C, D, E} and wherein: A -→ {B, C -» D} -→ E;

FIG. 9b is the SPE representation of the example shown in Figure 9a

FIG. 10 schematically illustrates a method for optimal expected runtime of a set of tests in a

cascade, in accordance with aspects of embodiments of the present invention, wherein the set of tests includes tests with ordering constraint in non-series-parallel structure, but with no statistical dependencies.

FIG. 11 schematically illustiates a method for optimal expected runtime of a set of tests in a cascade, in accordance with aspects of embodiments of the present invention, wherein the tests are statistically dependent only in pairs.

FIG. 12 schematically illustrates a method for optimal expected runtime of a set of tests in a cascade, in accordance with aspects of embodiments of the present invention, wherein the tests are statistically dependent, and an explicit distribution is provided.

FIG. 13 schematically illustrates a method for optimal expected runtime of a set of tests in a cascade, in accordance with aspects of embodiments of the present invention, wherein the tests are statistically dependent, and a training set of examples is provided.

DETAILED DESCRIPTION

[0025] Aspects of the present invention now will be described more fully hereinafter with reference to the accompanying drawings, in which variations and aspects of the present invention are shown. Aspects of the present invention may, however, be embodied in many different forms and should not be construed as limited to the variations set forth herein; rather, these variations are provided so that this disclosure will be thorough and complete, and will fully convey the scope thereof to those skilled in the art.

[0026] Unless otherwise defined, all technical and scientific terms used herein have the same meaning as commonly understood by one of ordinary skill in the art to which aspects of the present invention belong. The methods and examples provided heiein are illustrative only and not intended to be limiting.

[0027] By way of introduction, aspects of the present invention include providing methods of optimizing the cost of executing a set of tests, wherein optimizing the cost of executing the set of tests includes finding the optimal ordering the tests for some important cases, and near-optimal orderings for the rest of the cases, such that the resources required for executing the tests are minimized.

First embodiment: statistical independence and no ordering constraints

[0028] In accordance with aspects a first embodiment of the present invention, a method

is provided for an optimal ordering of tests in a cascade, whereas there are no pre defined order constraints between tests and no statistical dependencies between tests.

[0029] q, denote the tests qualities. Intuitively, quality represents the fraction of objects rejected by the test per unit time. It is thus very intuitive that tests be ordered by this quality measure:

Given a set of tests $x = \{x_1, x_2, \ldots, x_n\}$, the tests being both statistically independent and have no ordering constraints, ordering the tests in decreasing quality results in a minimal expected runtime of set x. Following Equation (2), the problem is reduced to finding the permutation O that minimizes the expected runtime:

$$ET(P) = \sum_{i=1}^{n} t_{O_i} \prod_{j=1}^{i-1} (1 - r_{O_j}) \quad (4)$$

[0030] Reference is now made to Figure 2, which illustrates method **100** for optimal expected runtime of a set of tests in a cascade, in accordance with aspects of embodiments of the present invention, wherein the set of tests includes independent tests with no statistical dependencies and no ordering constraints. Method **100** includes the following steps:

Step 110: Compute the quality value of test x,: $q_i = \frac{r_i}{t_i}$, where r_i denotes the marginal probability that test x_i rejects examples and t_i denotes the execution time of test x_i, or any other cost of executing test x_i.

Step 120: Schedule the tests commencing with the highest q_i, in decreasing order of q_i.

[0031] Example 1: if two tests x_i and x, , with $q_i < q_j$, are independent and x_j immediately follows x_i, then by exchanging x_i and x, we reduce the expected runtime of the sequence.

[0032] Example 2: suppose X = {A, B, C}, where:

- A: has a reject probability $r_A = 0.9$ and an execution time of $t_A = 10$;
- B: has a reject probability $r_B = 0.5$ and an execution time of $t_B = 1$; and
- C: has a reject probability $r_C = 0.01$ and an execution time of $t_C = 0.5$.

Hence:

q_A=0.09;

$q_B = 0.5$ and

qc = 0.02.

Since $q_B > q_A > q_C$, the optimal order is: *B, A, C*.

The expected runtime of each sequence, as computed by equation (4), is then:

ET(BAC) = 1+0.5*00+0.1*0.5) = 6.025;

ET(ABC) = 10+0.1 *(l+0.5*0.01) = 10.1005; and

ET(CBA) = 0.5+0.99*(l+0.5* **10)** = 6.44.

Second embodiment: statistical independence and series-parallel ordering constraints

[0033] The ordering problem with constraints (prerequisites) is an NP-hard (nondeterministic polynomial time) problem. However, for a rather extensive class of partial orders, hereinafter referred to as "series-parallel" structures (similar to the well-known series-parallel graphs), the present invention provides a provably optimal polynomial-time algorithm.

[0034] To be a series-parallel structure, the partial order must be specifiable using set construction "parallel" operator {} and the "series" operator -→ using a read-once expression over the tests.

[0035] The semantics of the operators is as follows:

- {A, B} groups the *A* and *B* test sets and ordering constraints, not adding any constraints. The parallel operator is commutative and associative.
- A -→ B groups the *A* and *B* test sets, adding the requirement that all tests in *A* must appear before all tests in *B*. The series operator is associative.

[0036] Reference is now made to Figure 3, which illustrates a standard graph representation of the following example of tests with structural (ordering) constraints is shown: X = {A -→ {M, N} -> B}, meaning that *B* succeeds both *M* and *N*, both of which must succeed *A*.

[0037] It should be noted that due to the read-once requirement, the sets of tests A and B are disjoint. The operator notation is extended to allow grouping of several sets of tests. For example, it is allowed to write {A. B, C} instead of {{A, B}. C} and A -→ B -> C instead of A -→ (B -> C).

[0038] More examples of series-parallel expressions over tests *a, b, c, x, y, z:*

(a) a->b->c-> χ->y->z defines the total order *abcxyz.*

(b) {a, b, c, x, y, z} defines a completely unconstrained [illegible]

(c) {a, b, c}-→{x, y, z} defines the K3,3 construct, where a, b, c must all precede x, y, z.

(d) a -→ {b -→ c, x -→ y} -→ z defines a construct diverging at a, and merging at z. Figure 4 illustrates the graph representation of this example.

(e) a -→ {b -→ c, x -> {y, z}} defines a forward branching tree with a as root.

[0039] It should be noted that series-parallel constructs include all partial orders whose structure graph is a tree, as well as all series-parallel directed graphs.

[0040] It should be noted that if an arbitrary finite number of dummy tests (that do not affect the partial order with respect to existing tests) are allowed to be added, thereby enhancing the representational power of series-parallel directed graphs, then series-parallel constructs would be exactly equivalent to series-parallel graphs. For instance, the K3,3 directed construct in Example (c) hereinabove is not a series-parallel graph. However, adding dummy tests dj, d_2, d_3, with precedence di -> (a, b, c} -> d_2 -> {x, y, z} -> d_3, we get a traditional series-parallel graph structure. Figure 5 illustrates the graph representation of this example.

[0041] It should be noted that the exemplary expression: {b -> x, {a, b} -→ y} is not series-parallel, as the expression is not read-once, having b appearing twice. Hence, polytree structured partial orders are not, in general, series-parallel structures.

[0042] Reference is now made to Figure 6, which illustrates recursive method 200 for optimal expected runtime of a set of tests in a cascade, in accordance with aspects of embodiments of the present invention, wherein the set of tests includes independent tests with no statistical dependencies but with ordering constraints.

[0043] Method 200 assumes that the tests are statistically independent, but have non-empty ordering constraints, which obey the series-parallel structure. The input to method 200 includes:

(a) A set of tests X, wherein each $x_i \in X$ includes a reject probability r, and a runtime t_i.

(b) A partial order, specified as a read-once series-parallel structure expression *(SPEJ)* which is processed as a lexical derivation tree. The internal nodes of this tree are labeled either S for a "series" constructor or P for a "parallel" constructor.

[0044] The output of method 200 is an optimal ordering of the tests.

[0045] Method 200 is a recursive method, including the following steps:

Step 210: ÏÏ *SPE* is not a leaf, go to step 230.

Step 220: Compute the quality value of the test x_t: $q_i = \frac{r_i}{t_i}$, where r_i denotes the marginal probability that test x_i is rejected and t_i denotes the execution time of test x_i or any other cost of executing testx,. This is the basic case where X includes only test X_1

Step 222: returns X (which is a single test x,) and exits.

Step 230: Denote ***R*** to **be** the root of ***SPE*** and ***CC*** to **be** the children of ***R*.**

Step 240: For each $C \in CC$ compute the optimal ordering of the tests *X(C),* applying method **200** to the tests *X(C).* with C as the SPE in the recursive call

Step 250: If *R* is a *P* operator, then apply Merge function **260** to the tests *X(C)* of each $C \in CC$ and go to step **290.**

The Merge function **(260)** gets one or more sorted sequences of tests as the input, and performs a merge by sorting together the tests in all the *X(C)* in decreasing quality <?,,, and returns the resulting sequence, being in optimal order.

Step 270: Else *R* being an *S* operator, concatenate all the tests in all of the *X(C)* in the order of appearance in *SPE,* thereby creating a list *L(C)* of tests.

Step 280: Conglomerate adjacent tests x,, x_{1+} ι in *L(C)* whenever q(x,) < q(x,+i), until no such cases exist.

The conglomeration process (illustrated in Figure 7) traverses the sequence of tests x, in *L(C),* according to the following scheme:

Step 282: Succeeding tests x_i, x_{i+1} are conglomerated if $q_i < q_{i+1}$, wherein the result is a new test *x',* that consists of both x_i and x_{i+1}. The new test x', replaces both x, and x_{i+1} in *L(C).* A record is kept of the fact that x', is a conglomeration of x,, x_{i+1}, along with the new ordered list.

Step 284: When no more tests x, in ***L(C)*** can be conglomerated, the resulting sequence, denoted by X, is returned.

Step 290: When nesting is completed the last returned X is returned as the optimal list of tests A'.

[0046] Example for method **200:**

[0047] Referring now to Figure 9a, a series-parallel structure of tests X is shown,

representation if the example shown in Figure 9a. Going through the nesting patli of method **200,** the process reaches the bottom tree level consisting of the singleton list of tests C. The quality of C is computed and returned. Likewise for D. At a higher level, we have the list of tests *L(C)* containing tests *C* and *D*, whereas tests *C* and *D* are in series (meaning ordering constraint: C -> *D).* Method **200** sets the order between tests C and *D* according to the SPE, conglomerating tests if necessary in step **280.** If for example qc < qo, then C and D are conglomerated into a single test CD, and we have X=(CD) going into the next upper level.

[0048] The current level of the SPE tree consists of the optimally ordered singleton list X=(CD) returned from the previous level and test *B,* being in parallel. Hence, method **200** calculates the quality of test *B* and merges **(step 260)** test *B* with *(CD),* forming an updated optimal ordered list *X,* which is returned to next level in the SPE tree. If for example $q_B < qc_D$, X-(CD, B) going into the next upper level.

[0049] The current level of the SPE tree consists of the list the optimal ordered list X=(CD, B) returned from the previous level and tests *A* and *E,* being in series. Method **200** calculates the quality of tests *A* and *E* (in lower recursive levels), and conglomerates **(step 280)** tests *A, {CD, B)* and *E,* thereby forming an updated optimal ordered list *X,* which is returned by method **200** as the final optimal ordered list *X*. If for example $q_A < qc_D$, conglomerating (A, CD) yields ACD. If $q_{ACD} > q_B < qe$) conglomerating B, E yields BE. Method **200,** in this example, returns: (ACD, BE).

[0050] An aspect of method **200** is to resolve "equivalent" (with respect to the constraints) tests. Referring back to Figure 3, a graph representation of the following example of tests, having structural constraints, is shown: X = {A -> (M, N } -> B). Tests *M* and *N* are said to be "equivalent" with respect to test *A,* and thereby tests *M* and *N* are ordered in order of the quality of tests *M* and *N*.

[0051] Reference is also made to Figure 8, which is a graph representation of the following example of structural constraints is shown: (A -> M -> B and A ⇝ N -> B and M -> N). In the example shown in Figure 3, *M* and *N* are considered "almost equivalent". In such a case, if the test with the lower quality is constrained to precede the test with the higher quality, then it must immediately precede it. The optimization problem is thus equivalent to one where the tests arc conglomerated. Thus, in this case if $q_M < q_N$, a conglomeration step is taken, where *M* and *N* are conglomerated into MV, resulting in: $q_M < q_{wN} < q_N$.

[0052] Example: suppose X = {A, B, C}, where:

- A: has a reject probability r_A = 0.9 and an execution time of t_A = 10;
- B: has a reject probability r_B = 0.5 and an execution time of t_B = 1; and
- C: has a reject probability re = 0.01 and an execution time of tc = 0.5.

Hence:

q_A=0.09;

qg = 0.5 and

q_c = 0.02.

Since $q_B > q_A > qc$, the optimal order is: *B, A, C,* when no constraints are set.

[0001] However, assume addition of the constraint: A -> B, with $q_B > q_A$.

We now conglomerate A and B, and the resulting reject probability of the conglomerated test AB is:

$$\Gamma(\Lambda B) = 1-(1-0.9)*(1-0.5)=0.95.$$

The expected runtime of *AB,* as computed by equation (4), is:

$$ET(AB) = 10+0.1*1 = 10.1.$$

Hence, the quality of conglomerated *test AB* is:

$$qAB = 0.95/10.1 = 0.094 \text{ (approx)}$$

Hence, the optimal ordering is: (AB)C.

[0053] Third embodiment: statistical independence and general ordering constraints Reference is now made to Figure 10, which schematically illustrates method **300** for locally optimal expected runtime of a set of tests in a cascade, in accordance with aspects of embodiments of the present invention, wherein the set of tests includes independent tests with general ordering constraints, not necessarily in a series-parallel structure. For example: the expression: K = {b -→ x, {a, b) -→ y} is not series-parallel, as the expression is not read-once, having *b* appearing twice.

[0054] Method **300** assumes that the tests are statistically independent, but have non-empty, general ordering constraints. The input to method **300** includes:

(a) An arbitrary set of tests X, wherein each $x, \in X$ includes a reject probability r, and a runtime t,.

(b) A partial order as defined by the general ordering constraints.

[0055] The output of method **300** is a locally optimal ordering of the tests.

[0056] Method **300** includes the following steps:

Step 310: For each pair of previously unchecked subsequent blocks A followed by B of tests (selected in step **311),** where a block is any contiguous subsequence of tests,

Step 312: Compute the quality value of block of tests A, denoted by q_A, and the quality value of block of tests B, denoted by q_B.

Step 314: If $q_A < q_B$ (and there is no constraint forbidding it), exchange A and B.

Step 320: If no further exchanges are possible, return the final list of reordered tests.

[0057] For a given a set of tests X, denote S be an arbitrary sequence composed of all the tests in X, and consistent with the prerequisite general ordering constraints. Any sequence S which method **300** cannot continue to improve is referred to as a local minimum. For series-parallel structures, a unique local minimum is guaranteed. But for any deviation from the series-parallel structure, such as exemplified by expression K, it is possible to generate a counterexample showing more than one local minimum.

[0058] Nevertheless, applying method **300** results in an approximation algorithm for the general case, including the given exemplary expression K.

Forth embodiment: statistically **dependent pairs and no ordering constraints**

[0059] Reference is now made to Figure 11, which schematically illustrates method **400** for optimal expected runtime of a set *oï* tests, in accordance with aspects of embodiments of the present invention, wherein the set of tests includes statistically dependent tests with but with no ordering constraints.

[0060] Method **400** assumes that the tests are statistically dependent in pairs (allowing also for tests which are completely independent, without loss of generality), but have no ordering constraints. The input to method **400** includes:

(a) An arbitrary set of tests X, wherein each $x, zX,$ possibly dependent in pairs, includes a reject probability r, and a runtime t, and the respective reject probability given the results of dependent tests

(b) At least one pair of tests $x_j, x, £ X$ having statistical dependency.

[0061] The output of method **400** is an optimal ordering of the tests where there exist statistical dependencies between pairs of tests in X.

[0062] Method **400** includes the following steps:

Step 410: For each dependent pair $x_{,,}$ x/selccted in step **411),** that has not yet been processed,

Step 412: Compute the quality of **x,** and $\boldsymbol{X_p}$

Step 414: Sort pair $x_{,,}$ X_j by quality value (and by reject value r, if equal in quality).

Step 416: Assuming, without loss of generality, that $q, >q,$ in X, introduce the respective ordering constraint x, -→ x, , and set r, = r_{jj}.

Step 420: Use method **200** (for series-parallel structures) to get the optimal ordering for **X.**

Fifth embodiment: general statistical dependence and optional ordering constraints

[0063] The algorithm uses $q_i = \frac{r_i}{t_i}$ values, but updates the rejection of each feature by taking into account that all previously executed tests have rejected.

[0064] Reference is now made to Figure 12, which schematically illustrates method **500** for optimal expected runtime of a set of tests in a cascade, in accordance with aspects of embodiments of the present invention, wherein the tests includes statistically dependent tests, and an explicit distribution is provided.

[0065] Method **500** updates the rejection probability of each test x, **e** X by taking into account that all previously executed tests (Step **510)** have rejected. The input to method **500** includes:

(a) An arbitrary set of tests X..

(b) A completely specified distribution R over the rejection probabilities.

[0066] Method **500** schedules the tests to be executed in an approximately optimal ordering of the tests where there exist statistical dependencies between of tests x € X defined

by R. If there are non-empty ordering constraints (as given by a partial ordering O), they are considered by the algorithm,

[0067] Method **500** includes the following steps:

Step 510: Compute (or re~compute, if this is not the first pass) marginal r_j values based on R, given that all previously scheduled tests decided "reject".

Step 520: Select $x_j \pounds X$ preferring highest $q_j = \frac{r_j}{c_j}$ from among all tests in X that may be scheduled so as to be consistent with O.

Finding the best q_j is performed by executing method **100** or any other method

Step 530: Schedule x_j as selected in step **520** to be executed next, and remove x_j from X.

Step 540: If tests remain to be scheduled, go to step **510.**

Otherwise the method ends.

[0068] In variations of the present invention, instead of a closed form version of distribution R, a set of training examples (data) and their classes are provided. In such a case, reject probabilities are estimated from the training data D.

[0069] Reference is now made to Figure 13, which schematically illustrates method **550** for approximately optimal expected runtime of a set of tests, in accordance with aspects of embodiments of the present invention, wherein the tests are statistically dependent, and a training set of examples is provided. Optionally, ordering constraints may be provided to the system.

[0070] Method **550** uses the training data D to estimate the reject probabilities (Step **560).** but updates the rejection probability of each test x, **e** X by taking into account previously executed tests through changing the set D accordingly. The input to method **550** includes:

(a) An arbitrary set of tests X.

(b) A training data set of examples D.

[0071] Method **550** schedules the tests to be executed in an approximately optimal ordering of the tests, where there exist statistical dependencies between of tests $x \in X$, with distributions estimated from D.

[0072] Method **550** includes the following steps:

Step 560: Estimate (or re-estimate, if this is not a first pass, and D is not too small, and in particular not empty) *r*, values based on the training data set *D*, by computing the test x, on all examples from *D*.

Finding the best q_i is performed by executing method **200** or in any other method.

Step 570: Select X_j **e** X such that $\frac{r_j}{t_j} \geq \frac{r_i}{t_i}$, Vx, **e** X and such that when scheduling x_j, it does not violate the ordering constraints O. That is, find the test that has the greatest quality $q_i = \frac{r_i}{t_i}$ by executing method **100** or any other method.

Step 580: Schedule x, as selected in step **570** to be executed next.

Step 582: Remove X_j from **X**.

Step 585: If no tests remain to be scheduled (X is empty), the method ends. Otherwise continue to step **590.**

Step **590:** Remove examples rejected by x, from Z), and go to step **560.**

[0073] In variations of the present invention, the "one sided perfect" assumption (the assumption that reject decisions made by tests arc always correct) is dropped. Instead, a false reject probability/ is defined for events when a tests *x*, incorrectly rejects a sample. The cost for false reject/ is C, and the quality of test X_p is:

$$q_i = \frac{r_i}{t_i - C \cdot f_i}$$

It should be noted that when $f_i = 0$, we have **[0001]**

[0074] Aspects of the present invention being thus described in terms of several variations and illustrative examples, it will be obvious that the same may be varied in many ways. Such variations are not to be regarded as a departure from the spirit and scope of the described aspects, and to incorporate such modifications as would be obvious to one skilled in the art.

CLAIMS:

1. A method for optimal ordering of tests in a set of tests, the method comprising:

(a) providing a set of tests X, wherein each $x_{,-} \in X$ includes the marginal reject probability r, that testx,- rejects and the cost t_i of executing test x,, wherein said tests $X_i \in X$ are statistically independent and wherein said tests $x_{,-} \in X$ have no ordering constraints;

(b) computing the quality value for each test X_p; and

(c) scheduling said tests by sorting $x_{,-} \in X$ in decreasing order of $q_{,}$.

2. The method, according to claim 1, wherein said cost t, of executing a test x, is the execution time of said test $x_{,-}$.

3. The method, according to claim 1, wherein said quality value for each testx,, is computed according to:

$$q_i = \frac{r_i}{t_i}.$$

4. The method, according to claim 1, wherein said quality value for each test $x_{,-}$, is computed according to:

$$q_i = \frac{r_i}{t_i + C * f_i},$$

Where f_i is the false reject probability and C is the cost tradeoff factor for the false rejections.

5. A method for optimal ordering of tests in a set of tests, the method comprising:

(a) providing a set of tests X, wherein each $x, \in X$ includes the marginal reject probability n that testx, rejects and the cost t_i of executing testx,-, wherein said tests $x, \in X$ are statistically independent; wherein at least a first test $x, \in X$ has an ordering constraint with respect to at least a second test $x, \in X$; and wherein the only type of ordering constraints allowed obey a series-parallel structure;

(b) providing a partial order, specified as a read-once series-parallel structure expression *(SPE)* having a lexical derivation tree structure, whereas the internal

nodes of said tree are labeled either with a "Series" constructor *S* or with a "Parallel" constructor *P;*

(c) processing said set of tests *X* thereby determining if said *SPE* contains a single test *x*, being a single leaf;

(d) when said *SPE* contains a single leaf, computing the quality value for each test *x*, and returning test x, and said computed quality;

(e) recursively processing said method for optimal ordering of tests in a set of tests *X(C),* where C e *CC, CC* denotes the children of *R* and *R* is the root of said *SPE;*

(f) when R is a *P* constructor, processing said tests *X(C)* with a merging process;

(g) when R is an *S* constructor, processing said tests *X(C)* with a conglomerating process; and

(h) scheduling said tests according to tests *x*, in resulting *X*.

6. The method, according to claim 5, wherein said quality value for each test *x,,* is computed according to:

$$q_{\iota} = \frac{r_{\iota}}{t_{,}}.$$

7. The method, according to claim 5, wherein said quality value for each test *x,,* is computed according to:

$$q_{\iota} = \frac{r_{:}}{t_{l+}c^{*}f_{\iota}},$$

where f_t is the false reject probability and C is the cost tradeoff factor for the false rejections.

8. The method, according to claim 5, wherein said (f) merging process of tests *X(C),* comprises the steps of:

i. providing one or more sorted sequences *X(C);* and

ii. sorting together all said tests *X(C)* in decreasing value of quality *q,*.

9. The method, according to claim 5, wherein said (g) conglomerating process of tests *X(C),* comprises the steps of:

i. concatenating all the tests *x*, in all of said *X(C),* in the order of appearance in said *SPE,* thereby creating a list of tests *L(C);*

ii. for all *x*, e *L(C),* when the quality of succeeding tests *x*, and $x_{,+\iota}$ fulfill $q, < q_{,+/}$,

conglomerating tests x, and $x_{,+I,}$ thereby creating a new test x', that consists of both x_t and x,+;; and

iii. replacing both x, and $x_{,+1}$ by said new test x', in $L(C)$.

10. The method, according to claim 9, wherein said conglomerating process further comprising the step of:

iv. recording the fact that x', is a conglomeration of X_{1}, x_{1+} i, of $L(C)$.

11. The method, according to claim 5. wherein said cost t, of executing a test x, is the execution time of said test x,.

12. A method for locally optimal ordering of tests in a set of tests, the set having cascade architecture, the method compiising:

(a) providing a set of tests X, wherein each x, c X includes the marginal reject probability r, that test x, rejects and the cost t, of executing testx,, wherein said tests x, € Xare statistically independent, and wherein at least a first tests X_1 e X has an ordering constraint with respect to at least a second tests x_j e X;

(b) providing a partial order O;

(c) processing said set of tests X thereby determining a block A of tests in X can be exchanged with subsequent block B of tests in X, while obeying said order O;

(d) for all subsequent blocks A and B of tests in X that can be exchanged, providing obeying said order O, computing the quality of blocks A and B by:

$$q_A = \frac{r_A}{t_A} \quad \text{and} \quad q_B = \frac{r_B}{t_B};$$

(e) when $qA < q_B$, exchange blocks A and B in X.

13. The method, according to claim 12, wherein said cost t, of executing a test x, is the execution time of said test X_1.

14. A method for optimal ordering of tests in a set of tests, the method comprising:

(a) providing a set of tests X, wherein each x, e X includes the marginal reject probability r, that testx, rejects and the cost i, of executing testx,, wherein there exists at least one pair of tests x, e X and x_j e X, whereas test x, is statistically dependent on a test x,;

(b) providing said statistical dependency;

(c) for each pair of tests x_i and x_j computing the quality of tests x_i and x_j by:

$$q_i = \frac{r_i}{t_i} \quad \text{and} \quad q_j = \frac{r_j}{t_j};$$

(d) sorting pairs x_i and X_j by said computed quality;

(e) when x_i has higher quality than x_j in X:

i. introducing an ordeiing constraint $x_i \rightarrow x_j$, and

ii. setting: $r_j = r_{j|i}$.

(f) when X_j has higher quality than x_i in X:

i. introducing an ordering constraint $x_j \rightarrow x_i$, and

ii. setting: $r_i = r_{i|j}$.

(g) performing the method for optimal ordering of tests in a set of tests, according to claim 5.

15. The method, according to claim 14, wherein said cost t_i of executing a test x_i is the execution time of said test x_i.

16. A method for approximate optimal ordering of tests in a set of tests, the method comprising:

(a) providing a set of tests X, wherein each $x_i \in X$ includes the cost t_i of executing test x_i, having an optional ordering constraint O;

(b) providing a distribution R over said rejection probabilities r of all $x \in X$;

(c) using said distribution R for re-computing marginal reject probability r_i values based on said distribution R given the condition that all previously removed tests have rejected.

(d) selecting $X_j \in X$ such that $\frac{r_j}{t_j} \geq \frac{r_i}{t_i}$, $\forall x_i \in X$ and such that X_1 can be scheduled according to said ordering constraint O;

(e) scheduling said selected X_j to be executed;

(T) Removing said X_1 from X

17. The method, according to claim 16, wherein said cost t_i of executing a test x_t is the execution time of said test x_i.

18. The method, according to claim 16, wherein said quality value for each test x_i, is computed according to:

$$q_i = \frac{r_i}{t_i}.$$

19. The method, according to claim 16, wherein said quality value for each test x_i, is computed according to:

$$q_i = \frac{r_i}{ti + C * f_i},$$

Where f_i is the false reject probability and C is the cost tradeoff factor for the false rejections.

20. A method for approximately optimal ordering of tests in a set of tests, the method comprising:

(a) providing a set of tests Jf, wherein each $x_i \in X$ includes the marginal reject probability r_i that test x_i rejects and the cost t_i of executing test x_i, having an optional ordering constraint O;

(b) providing a training data set of examples D;

(c) initializing and re-computing said marginal reject probability r_i values based on said training data set D, by computing the testx, on all examples from said training data set D;

(d) selecting $x_j \in X$ such that $\frac{r_j}{t_j} \geq \frac{r_i}{t_i}$, $\forall x_i \in X$, and such that X_j can be scheduled according to said ordering constraint O;

(e) scheduling said selected x_j to be executed;

(f) removing examples rejected by X_1 from said training data set D.

21. The method, according to claim 20, wherein said cost t_i of executing a test x_i is the execution time of said testx,.

22. The method, according to claim 20, wherein said quality value for each testx,, is computed according (o:

$$q_1 = \frac{r_i}{t_i}.$$

23. The method, according to claim 20, wherein said quality value for each test x_i, is computed according to:

$$q_i = \frac{r_i}{t_i + C * f_i},$$

Where f_i is the false reject probability and C is the cost tradeoff factor for the false rejections.

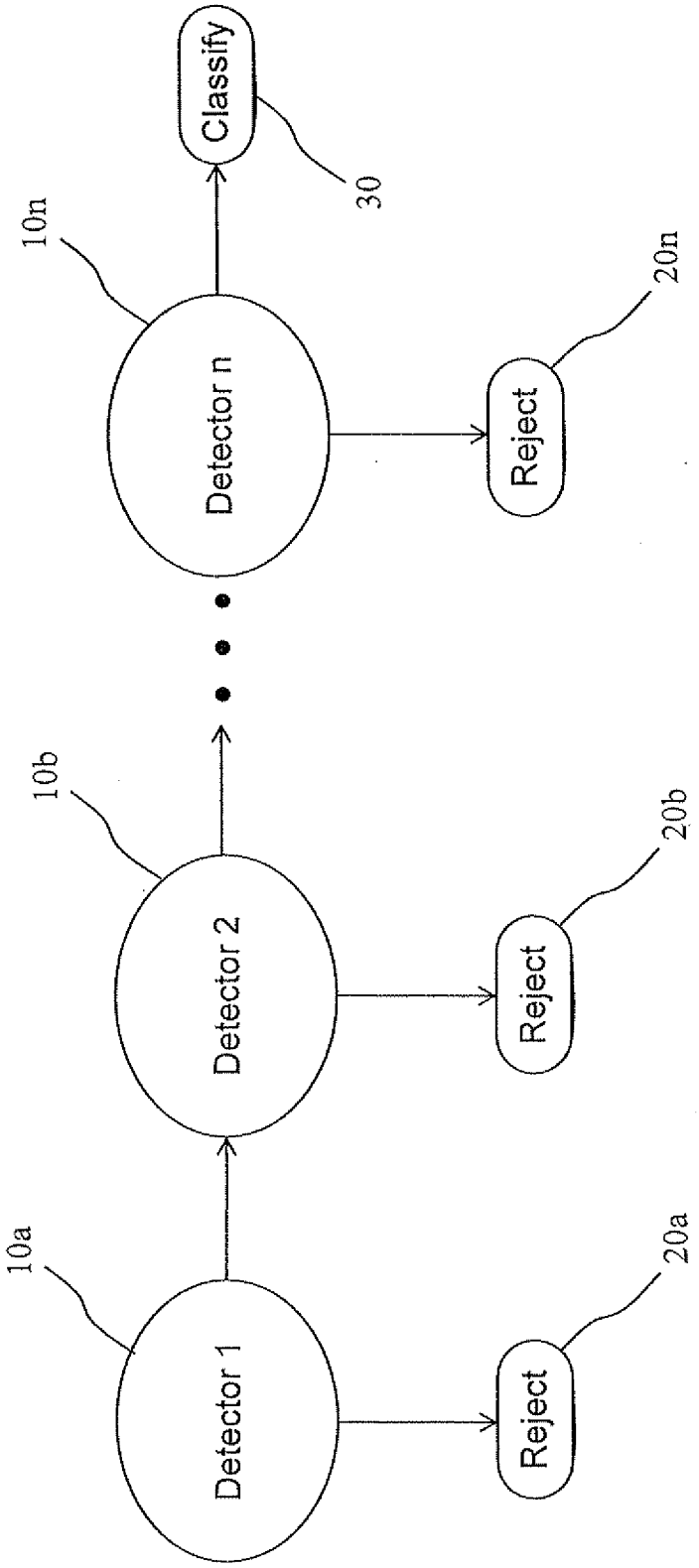

Fig. 1

PRIOR ART

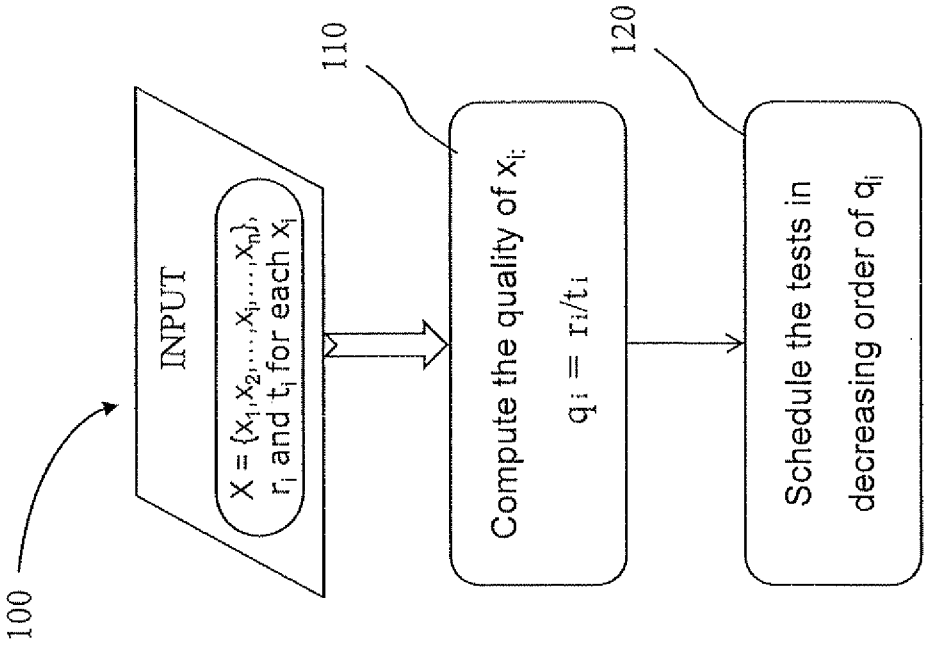
100
INPUT
X = {x1,x2,...,xi,...,xn},
ri and ti for each xi
110
Compute the quality of xi:
qi = ri/ti
120
Schedule the tests in
decreasing order of qi

Fig. 2

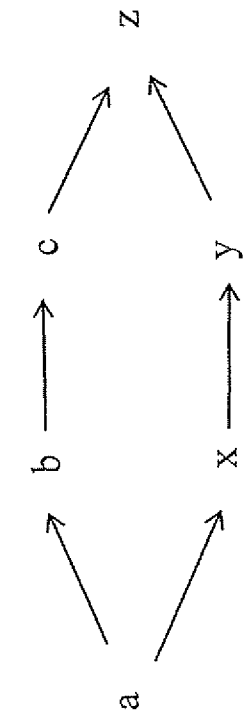

Fig. 4

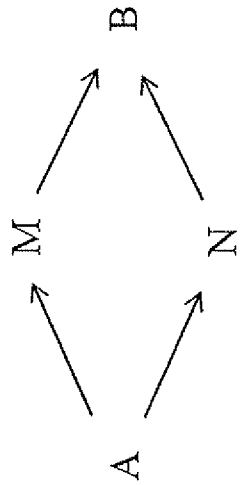

Fig. 3

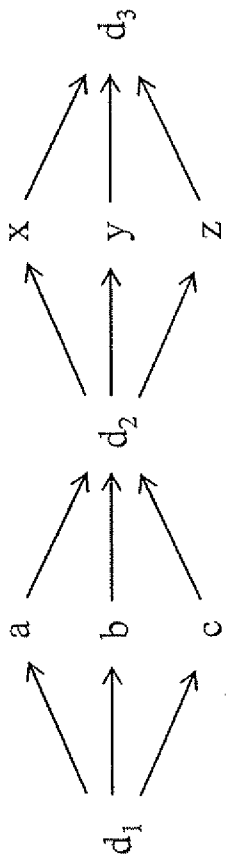

Fig. 5

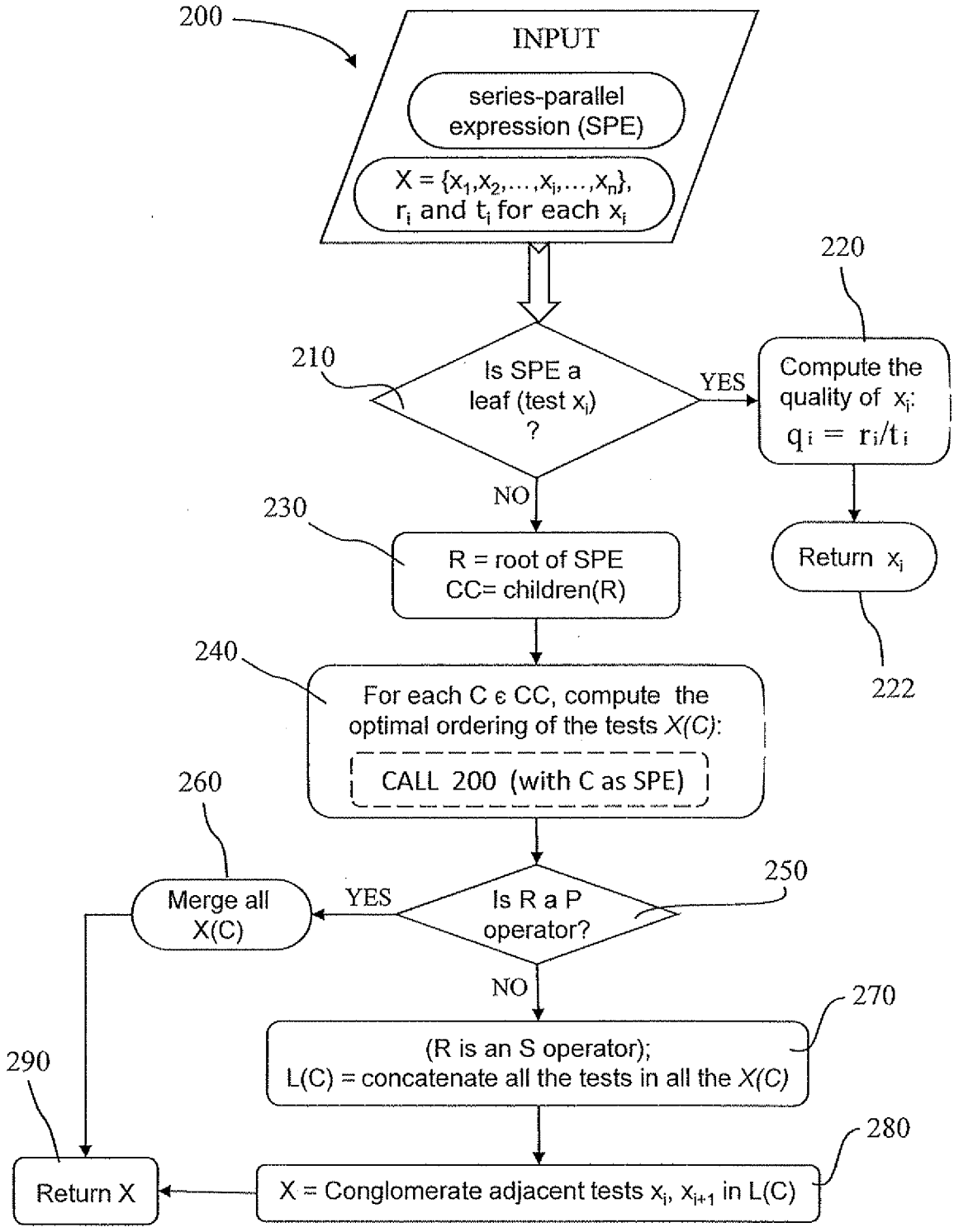
200
INPUT
series-parallel expression (SPE)
X = {x1,x2,...,xi,...,xn}, ri and ti for each xi
210
Is SPE a leaf (test xi) ?
YES
220
Compute the quality of xi: qi = ri/ti
Return xi
222
NO
230
R = root of SPE
CC= children(R)
240
For each C ε CC, compute the optimal ordering of the tests X(C):
CALL 200 (with C as SPE)
250
Is R a P operator?
YES
260
Merge all X(C)
NO
270
(R is an S operator);
L(C) = concatenate all the tests in all the X(C)
280
X = Conglomerate adjacent tests xi, xi+1 in L(C)
290
Return X

Fig. 6

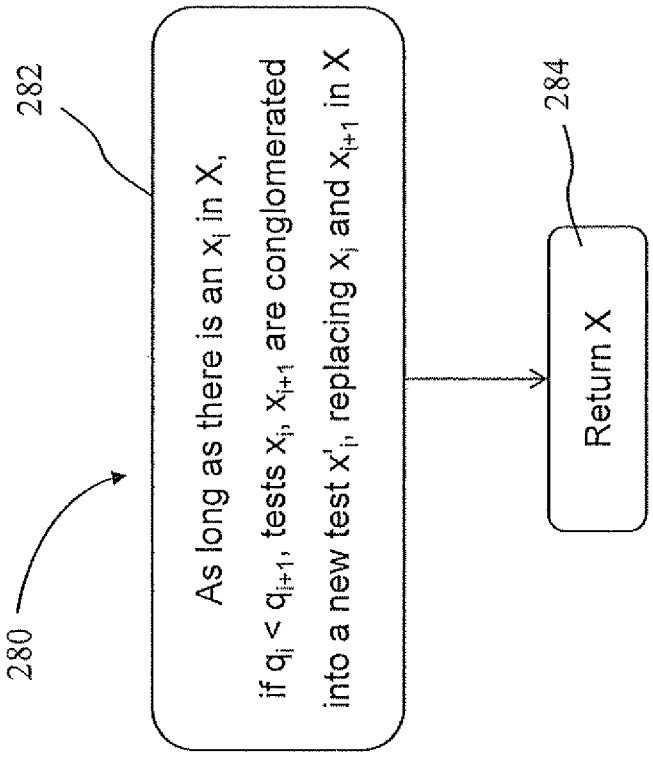
280
282
As long as there is an x_i in X,
if $q_i < q_{i+1}$, tests x_i, x_{i+1} are conglomerated
into a new test x'_i, replacing x_i and x_{i+1} in X
284
Return X

Fig. 7

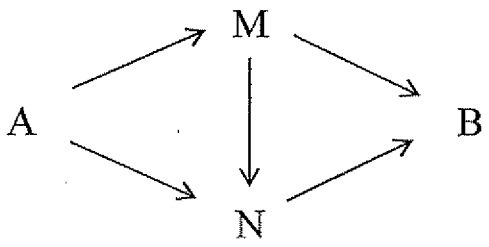

Fig. 8

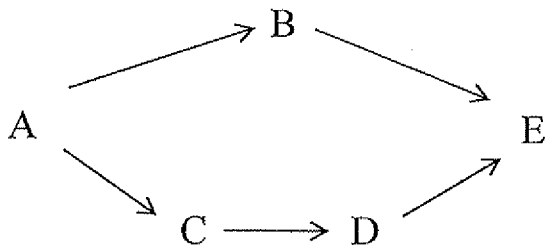

Fig. 9a

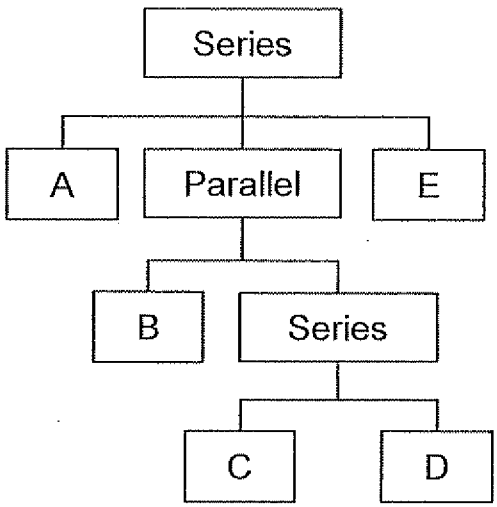

Fig. 9b

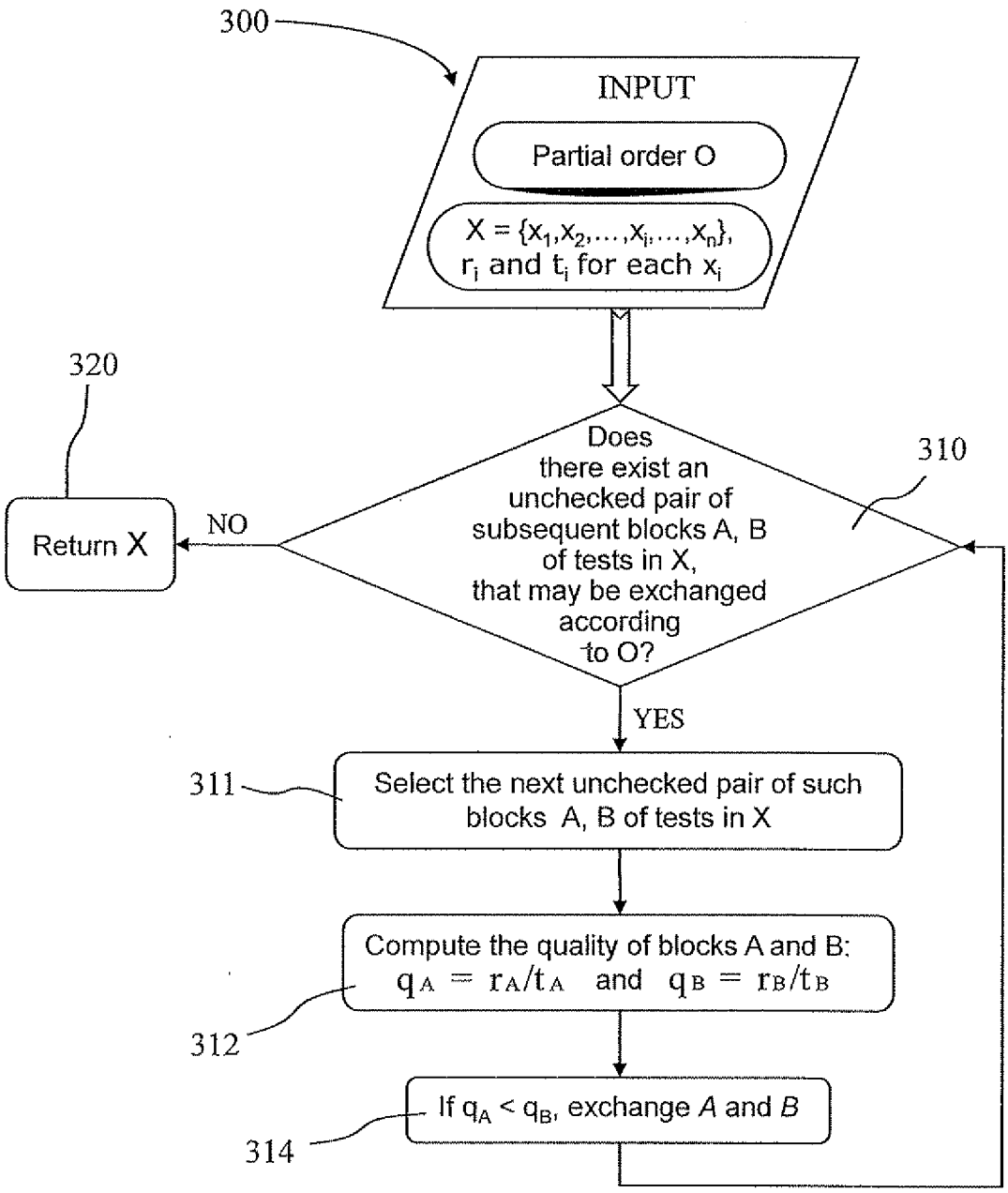
300
INPUT
Partial order O
X = {x1,x2,...,xi,...,xn},
ri and ti for each xi
320
Return X
NO
Does
there exist an
unchecked pair of
subsequent blocks A, B
of tests in X,
that may be exchanged
according
to O?
310
YES
311
Select the next unchecked pair of such
blocks A, B of tests in X
Compute the quality of blocks A and B:
qA = rA/tA and qB = rB/tB
312
If qA < qB, exchange A and B
314

Fig. 10

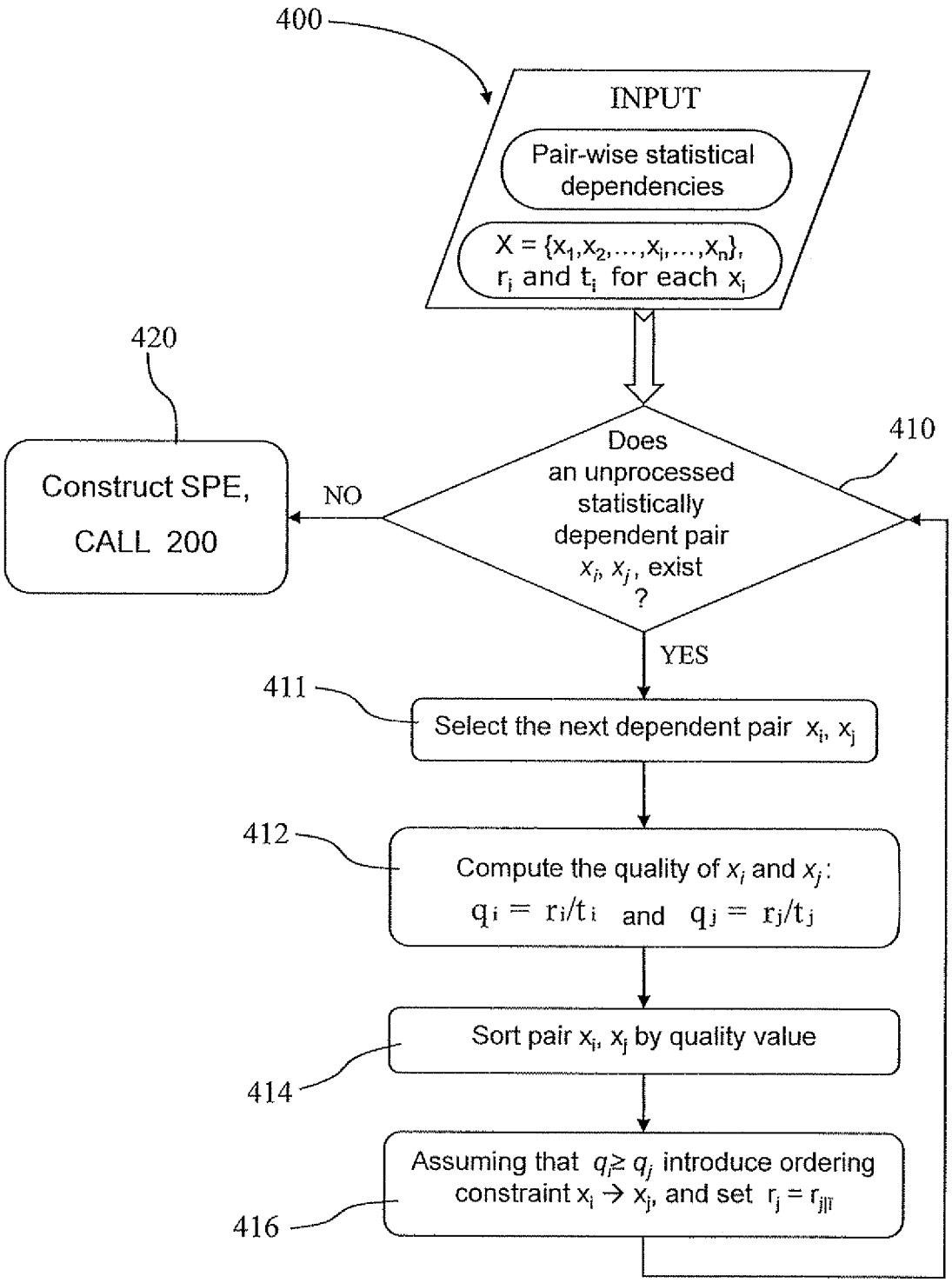
400
INPUT
Pair-wise statistical dependencies
X = {x1,x2,...,xi,...,xn}, ri and ti for each xi
420
Construct SPE, CALL 200
NO
Does an unprocessed statistically dependent pair xi, xj, exist ?
410
YES
411
Select the next dependent pair xi, xj
412
Compute the quality of xi and xj: qi = ri/ti and qj = rj/tj
414
Sort pair xi, xj by quality value
416
Assuming that qi≥ qj introduce ordering constraint xi → xj, and set rj = rjj

Fig. 11

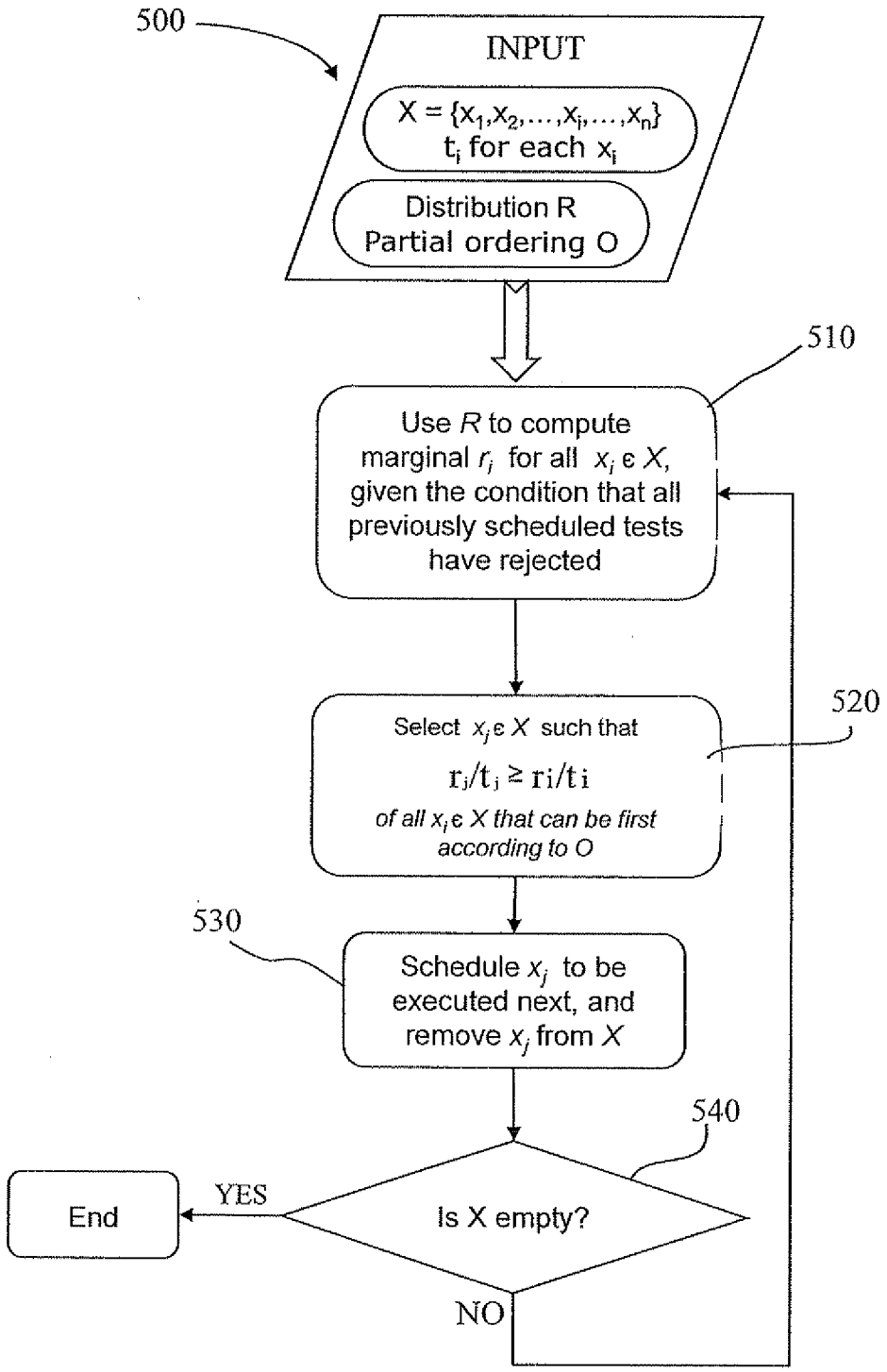
500
INPUT
X = {x1,x2,...,xi,...,xn}
ti for each xi
Distribution R
Partial ordering O
510
Use R to compute marginal ri for all xi ε X, given the condition that all previously scheduled tests have rejected
520
Select xj ε X such that
rj/tj ≥ ri/ti
of all xi ε X that can be first according to O
530
Schedule xj to be executed next, and remove xj from X
540
Is X empty?
YES
End
NO

Fig. 12

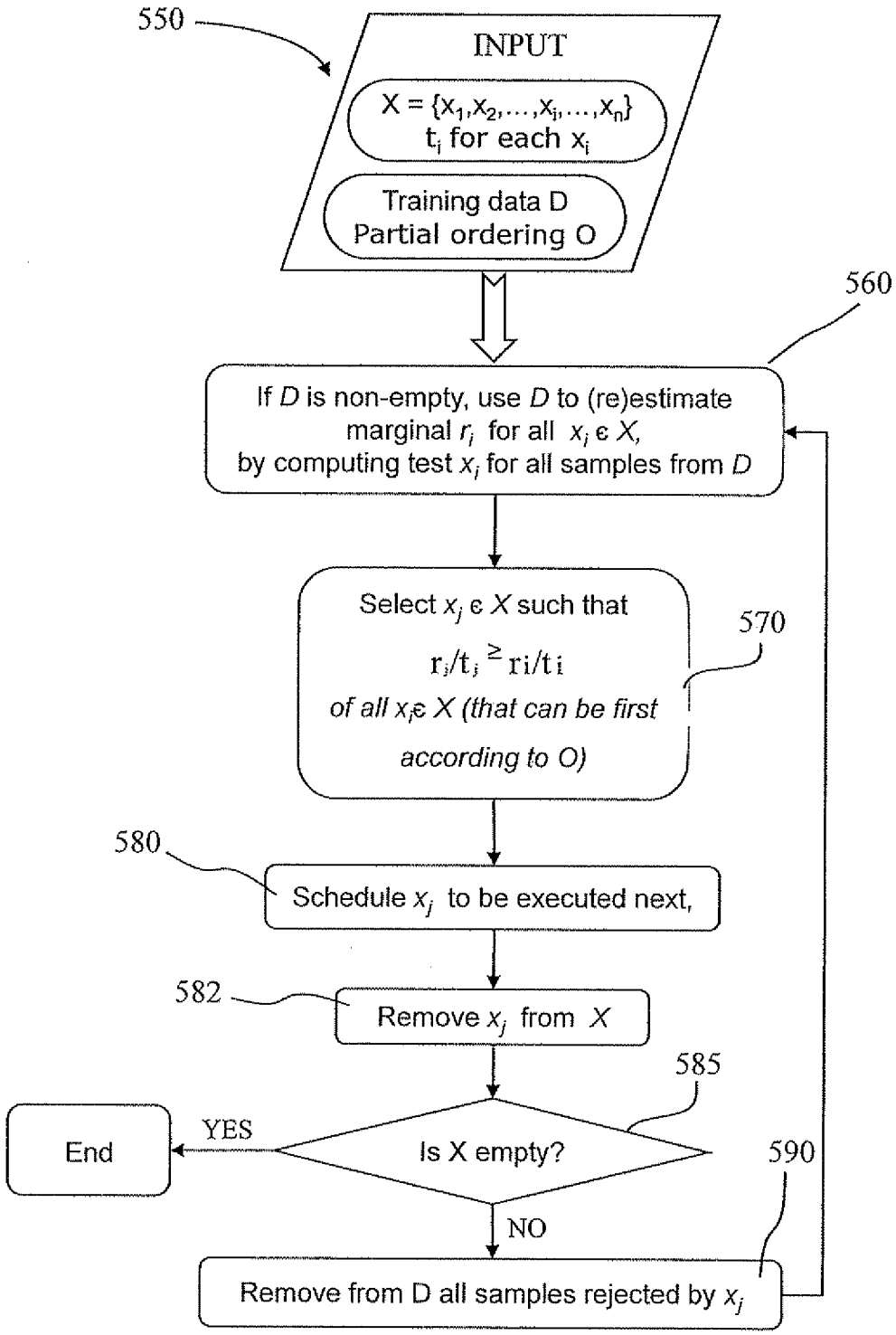
550
INPUT
X = {x1,x2,...,xi,...,xn}
ti for each xi
Training data D
Partial ordering O
560
If D is non-empty, use D to (re)estimate marginal ri for all xi ϵ X, by computing test xi for all samples from D
Select xj ϵ X such that
rj/tj ≥ ri/ti
of all xiϵ X (that can be first according to O)
570
580
Schedule xj to be executed next,
582
Remove xj from X
585
End
YES
Is X empty?
590
NO
Remove from D all samples rejected by xj

Fig. 13