



Measuring Progress in Premo Order Verification

Curtis Ober*

Exploratory Simulation Technology

Ryan Bond

Aerosciences

Patrick Knupp

Optimization and Uncertainty Estimation

7th World Congress on Computational Mechanics

July 16 – 22, 2006, Los Angeles, California

Minisymposium: Accomplishments and Challenges in Verification & Validation



Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy under contract DE-AC04-94AL85000.





Motivation for a Practical Study

- Create an example on how to implement theory
 - This is **an example**, not a recipe.
 - Different codes may follow different processes
 - Provide some lessons learned
- Some other benefits
 - Identification of weaknesses and practical issues with theory
 - Progress measures for Premo verification
 - New order-of-accuracy tests (OATs) for Premo
- Premo was chosen because of its verification “maturity”
 - Roy, Smith, Ober, & Nelson, 2002-2004
steady MMS, 2D, Cartesian meshes, Euler & Navier-Stokes equations
 - Bond, Ober, & Knupp, 2004-present
steady MMS, 3D, curvilinear hex meshes, Euler, Navier-Stokes, & RANS equations, boundary conditions
 - Dimiduk & Ober, 2005
unsteady MES, various temporal integration schemes



Scope of Progress Study

- Premo 1.3 β - compressible fluid dynamics software
 - Static code (October 2005)
 - 326 files
 - 3600+ functions
 - 182 input keywords
 - 83,000+ executable lines
 - 230,000+ total lines
 - Code capabilities excluded from the study
 - Deprecated and emerging capabilities
 - SIERRA framework or third-party libraries (TPLs)
 - Not all combinations of code capabilities
 - Focused on (outline)
 - Construction of Premo order-verification test-suite (OVTS)
 - Calculating the progress and fitness measures for Premo
- } Statistics limited to Premo



OVTS Construction

OVTS construction is iterative:

- 1) Start with initial set of OATs.
- 2) Evaluate OVTS for coverage holes/gaps in OV-domain.
- 3) Introduce new OATs or change existing OATs to fill coverage holes/gaps.
- 4) Continue testing for completeness and filling holes until completeness requirements are met.

IC dependent
evaluator dependent
sequence dependent &
granularity dependent

Does not result in a unique OVTS.



OVTS Construction - Step 1)

1) Start with initial set of OATs.

– OATs included in OVTS

- 9 steady, MMS tests for spatial order verification of interior-equation sets and BCs
- 7 convecting-vortex tests (unsteady, inviscid, exact solutions) tests used for temporal-integration verification

– OATs excluded from OVTS

- Steady, 2D, Cartesian meshes, Euler and Navier-Stokes equations
- Reason: less general and covered redundant functionality
- Still useful; provides fine-grained coverage



OVTS Construction - Step 2)

- 2) Evaluate OVTS for coverage holes/gaps in OV-domain.
- Evaluators for identifying OVTS coverage holes
 - Expert knowledge - most general way of finding holes
 - Function coverage - based on code member functions
 - Keyword coverage - based on input parameter names, flag names,...
 - Line coverage - not well suited for order-of-accuracy analysis
 - Some holes could only be identified by **one** evaluator
 - Completeness requirement is the most difficult part
 - Need a mapping from governing equations/code capabilities to OV-domain/lines of code
 - There is still a need to develop methods to better facilitate code-coverage evaluation.

OVTS Construction - Step 2)

Evaluators	Pros	Cons
Expert knowledge*	<ul style="list-style-type: none">• Best for initial design• Catches higher concepts<ul style="list-style-type: none">– Combinations of capabilities– Are OATs general?	<ul style="list-style-type: none">• Not automated• Requires intimate knowledge of software and governing equation
Function coverage*	<ul style="list-style-type: none">• Traces code execution• Automated	<ul style="list-style-type: none">• Tested in most general way?• Misses finer-grained branches
Keyword coverage*	<ul style="list-style-type: none">• User-oriented measure• Automated	<ul style="list-style-type: none">• Tested in most general way?• Does not trace code execution
Line coverage	<ul style="list-style-type: none">• Finds all untested lines• Traces code execution• Automated	<ul style="list-style-type: none">• Finds all untested lines• Requires expert knowledge<ul style="list-style-type: none">– Lines are in the OV-domain– Connection between code capability and lines of code

* Need a variety of evaluators to thoroughly identify holes.

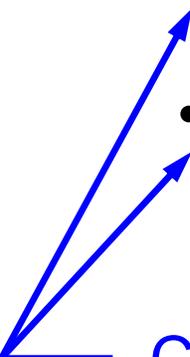


OVTS Construction - Step 3)

3) Introduce new OATs or change existing OATs to fill coverage holes/gaps.

– Types of tests

- *Exact solutions* can be used where they exist.
 - **Example:** Convecting vortex OATs tests temporal terms
- *Manufactured solutions* can be used to test features in the most general way.
 - **Example:** All spatial derivatives are generally tested
- *Unit tests* can be used to evaluate modular components
 - **Example:** viscosity models, $\mu=\mu(T)$
- Others
 - End-to-end functional tests



Can include non-OATs



OVTS Construction - Step 3)

- 3) Introduce new OATs or change existing OATs to fill coverage holes/gaps.
- How you add/change OATs determines the type of OVTS
 - *Many* OATs with *few* unique code capabilities leads to a **fine-grained** OVTS
 - Isolate coding mistakes and algorithmic weaknesses easier
 - *Few* OATs with *many* unique code capabilities leads to a **coarse-grained** OVTS
 - Fewer tests and less cost to run them
 - Obvious examples
 - Copying and modifying an existing OAT creates redundancy → **fine-grained OVTS**
 - Replacing an OAT with an OAT which has a superset of capabilities reduces redundancy → **coarse-grained OVTS**
 - Granularity affects progress and fitness measure



OVTS Construction - Step 4)

- 4) Continue testing for completeness and filling holes until *completeness requirements* are met.
 - Use coverage evaluators to assess completeness
 - If one can only construct redundant OATs, then finished.
 - Again, we did not try to create all combinations



OVTS Construction - Step 4)

- 4) Continue testing for completeness and filling holes until *completeness requirements* are met.
 - 52 total OATs in current Premo OVTS
 - 9 convecting vortex tests
 - 8 unsteady exact solution tests for limiter options
 - 35 MMS tests
 - Not all OATs are finished, but **all are defined**.
 - Required inputs are defined so that coverage evaluators can be used.
 - Full verification runs will be completed later (e.g., mesh refinement)
 - Once we have the OATs, we can **prioritize** them.

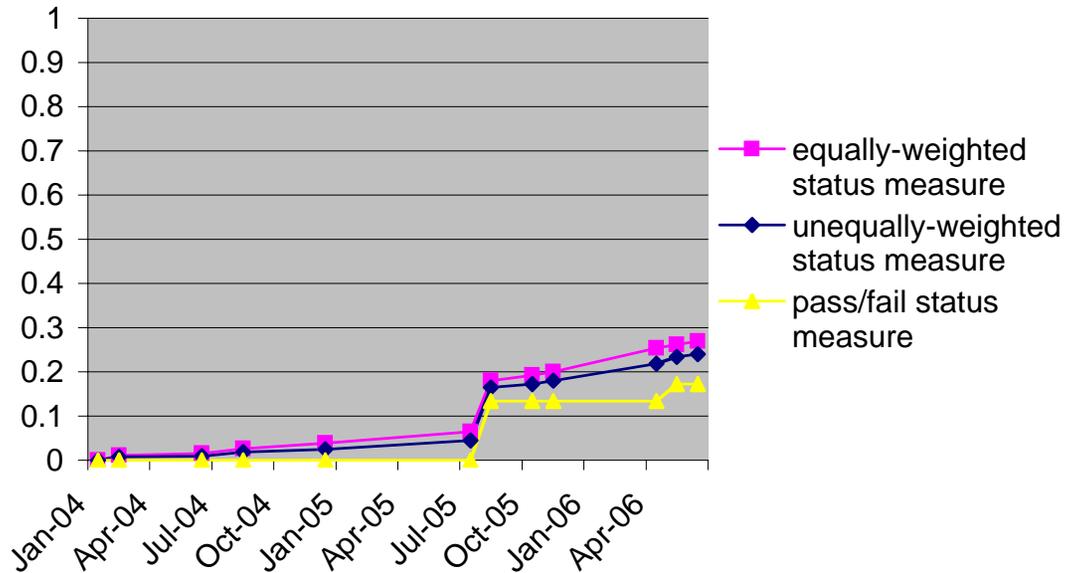
Progress Measures

Status-Based Progress Measure

indicates what fraction of the OATs are **passing** or are “**partially**” passing.

Process point-of-view

$$P_{\pi,1} = \frac{1}{N} \sum_{n=1}^N \frac{s(\ell_n)}{s(\ell_5)}$$



- Cheap to compute these measures

Level	Status	Equally-weighted	Unequally-weighted	Pass/Fail	Number of OATs
0	incomplete	0	0	0	23
1	ready	1	3	0	15
2	numerical solutions exist	2	5	0	5
3	all solutions asymptotic	3	10	0	0
4	all orders verified	4	12	0	0
5	OAT results reproducible	5	20	1	9

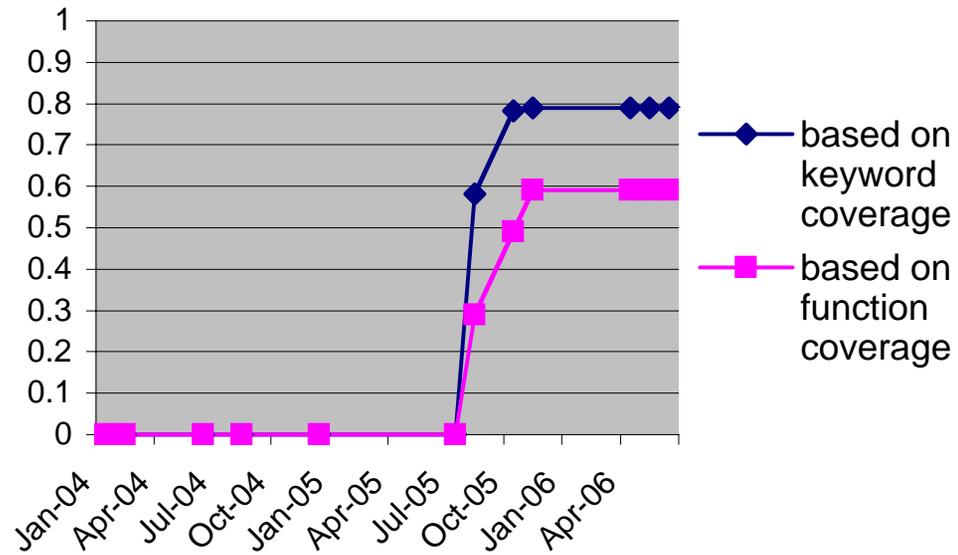
Progress Measures

Coverage-Based Progress Measure

indicates what fraction of the functions/keywords are **passing/verified**.

$$P_{\pi,3} = \frac{\# \text{ functions covered by passing OATs}}{\# \text{ functions covered by OVTS}}$$

$$P_{\pi,4} = \frac{\# \text{ keywords covered by passing OATs}}{\# \text{ keywords covered by OVTS}}$$



- $P_{\pi,1} \sim 20\%$ and $P_{\pi,3} \sim P_{\pi,4} \sim 70\%$
 - Tests cover a large fraction of functions and large overlaps exist between tests. \Rightarrow principle of diminishing returns
- Coverage-based progress measures still lack knowledge if OATs are general tests of functionality.
 - Thus may falsely inflate or deflate the significance of some tests.
- Trends are important, do not read too much into the numbers

Fitness Measures

Fitness measure

indicates what fraction of the capabilities for a given application has been verified.

$$F_{\pi,1} = \sum_{n=1}^N w_n r_n$$

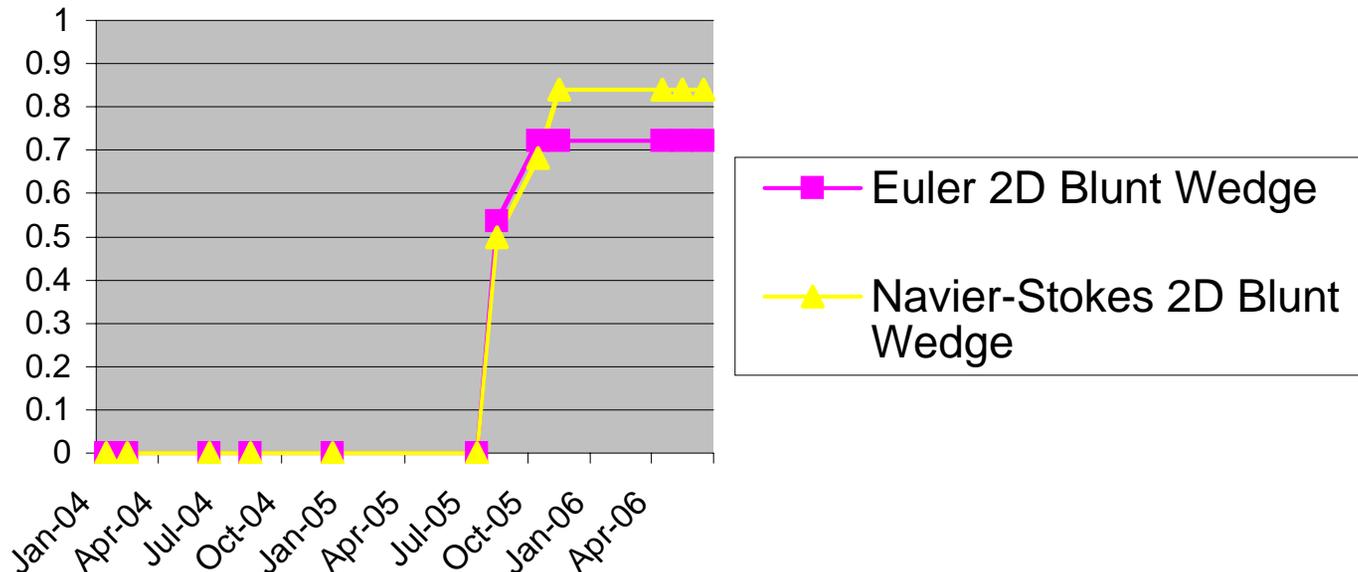
Number of relevant functions

Pass/fail status

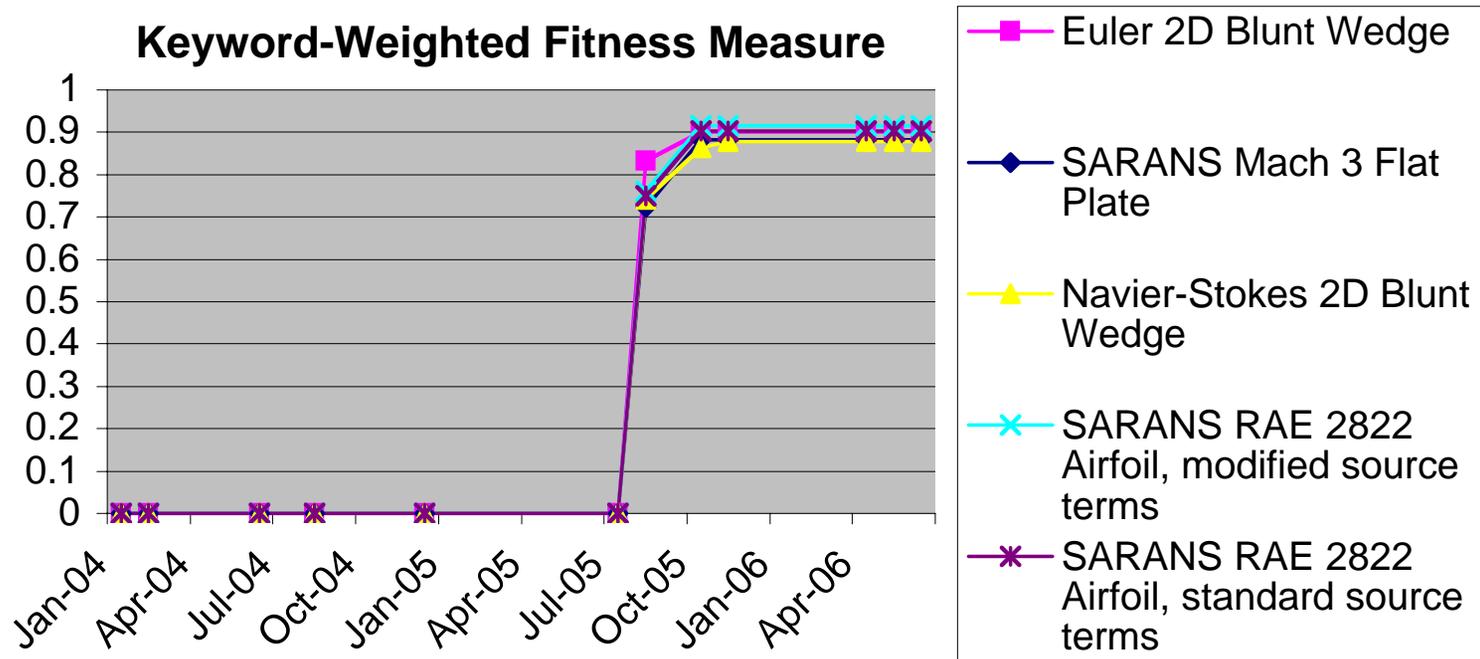
Function-based weighting

Usage point-of-view

Function-Weighted Fitness Measure



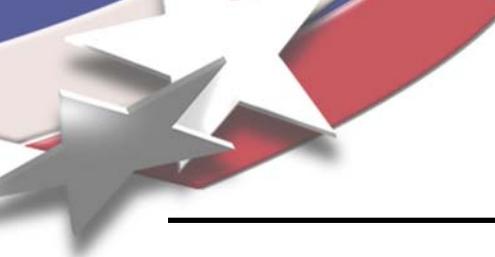
Fitness Measures



- Not at a 100% for these applications, but we have known OATs to help get there.
- What values are other codes getting for these fitness measures?

$$F_{\pi,1} = \sum_{n=1}^N w_n r_n$$

← **Number of relevant keywords**
← **Pass/fail status**
← **Keyword-based weighting**



Conclusions

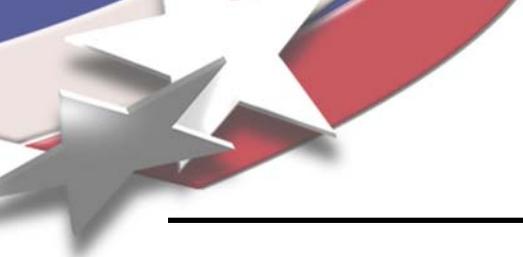
- Need a variety of evaluators to identify holes.
- OVTS will mostly be comprised of MES and MMS (OATs), but other non-OATs can be useful.
- Granularity of OVTS affects
 - progress and fitness measures
 - OVTS size and runtime
 - diagnostic effectiveness
- A constructed OVTS allows prioritization of OATs to be completed.
- Progress and fitness trends and relative values can be tracked, but do not read too much into the numbers.
- Computing progress and fitness measure is cheap, therefore include several measures.
- Premo not at 100%
 - But we know the OATs to get there
 - How does this compare with other codes?



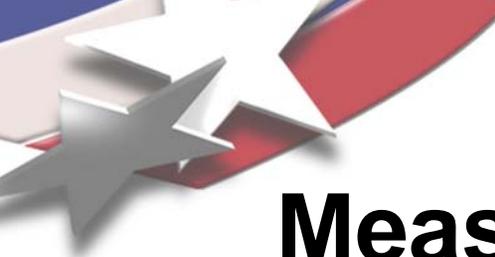
Acknowledgments

The authors would like to thank

- [Jeff Fisher](#) for his assistance in setting up some tests and modifying Premo and the SNTools to support needed coverage calculations.
- [Alfred Lorber](#) for his help with his function coverage evaluator.
- [Tolulope Okusanya](#) for his help with his keyword coverage evaluator.



Backup Slides



Measuring Progress in Premo Order Verification

14 June, 2006

Ryan Bond, 1515
Curt Ober, 1433
Pat Knupp, 1411



Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company,
for the United States Department of Energy under contract DE-AC04-94AL85000.





Outline

- Introduction
- Construction of Order Verification Test Suite (OVTS)
- Premo's OVTS
- Premo's measure history with this OVTS
- Conclusions
- Open Issues
- Acknowledgments



Definitions

- **Order-of-Accuracy Test (OAT):** a test intended to measure the order of accuracy (OA) for a given set of code inputs
- **Order Verification Test Suite (OVTS):** a collection of OATs associated with a particular version of a code
- **Order Verification Exercise (OVE):** the process of setting up & running an OAT, calculating the OA, and documenting the results
- **Discretization Algorithm:** any calculation with an associated OA or calculation which, if performed incorrectly, would adversely affect the observed OA



History of Premo Order Verification

- Premo was chosen because of the maturity of its verification efforts.
- Roy, Smith, Ober, & Nelson, 2002-2004
steady MMS, 2D, Cartesian meshes, Euler & Navier-Stokes equations
- Bond, Ober, & Knupp, 2004-present
steady MMS, 3D, curvilinear hex meshes, Euler, Navier-Stokes, & RANS equations, boundary conditions
- Dimiduk & Ober, 2005
unsteady MES, various temporal integration schemes
- To date, several coding mistakes have been found and corrected, other unresolved issues, probably related to algorithmic weaknesses, have been identified.



Motivation for Premo Study

- We need to apply the theory to a production code to identify weaknesses, gaps, and practical issues.
- We need to develop and test different ways of evaluating OVTS completeness.
- We need to establish a database of results for further investigation on progress and order verification.
- Create an example for how to implement progress measure theory, different codes may follow slightly different processes – an example, not a recipe.



Scope of Premo Progress Study

- Looking at order-verification versus other types of code verification.
- Focus is on test development and demonstration versus regression testing.
- version 1.3 of Premo (static code & OVTS)
- The scope of code capabilities tested was limited to allow useful results to be obtained in FY06:
 - no deprecated or emerging capabilities
 - no attempt to cover Sierra or TPL option spaces



Process for Constructing OVTS

OVTS construction is iterative:

IC dependent 1) Start with some incomplete suite of OATs

evaluator dependent 2) Evaluate this initial test suite for coverage holes.

sequence dependent & granularity dependent 3) Introduce new OATs or change existing OATs to fill coverage holes.

4) Continue testing for completeness and filling holes until completeness requirements are met

Does not result in a unique OVTS.



Ways of Identifying OVTS Coverage Gaps

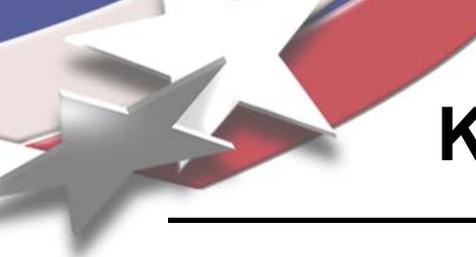
- **Expert knowledge:** start from a comprehensive list of discretization algorithms; then determine whether or not they are all covered in the most general fashion.
- **Function coverage:** identify a set of functions which should be touched by at least one OAT in the OVTS; then use function tracing to evaluate completeness.
- **Keyword coverage:** identify a set of input file keywords that should be exercised; then analyze the code input for each OAT to evaluate completeness.
- **Line coverage:** better suited for other types of testing than OA analysis.

These coverage evaluators identify gaps in particular equations, auxiliary equations, numerical methods, shape functions, etc. **We have not yet focused on combinations.**



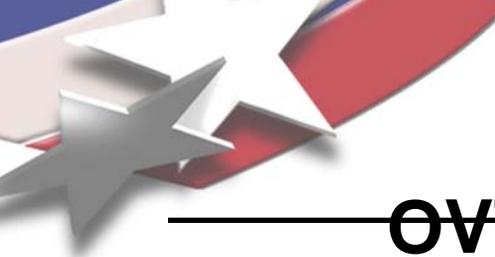
Function Coverage Evaluator

- Start with all ‘apply’ and ‘execute’ functions.
- Eliminate functions associated with deprecated or emerging capabilities.
- Run all tests in OVTS with function tracing.
- Compile statistics to see how many OATs call each function.
- Functions not called by any OATs represent holes (but other holes may still exist).



Keyword Coverage Evaluator

- Identify valid input file keywords.
- Filter out keywords valid for Sierra & TPL's but not related to Premo's order verification.
- For keywords associated with enumerated options (e.g., flux functions), make sure each option is covered.
- For keywords associated with floating point values (e.g., Prandtl number), we just make sure at least one test includes the keyword.
- Evaluate OVTS by parsing input files for keywords.
- Missing keywords or associated options represent holes (but other holes may still exist).



Pros & Cons of Different OVTS Completeness Evaluators

Method	Pros	Cons
Expert knowledge	<ul style="list-style-type: none">•best for initial design•catches higher concepts missed by automated methods	<ul style="list-style-type: none">•not automated•requires adequate documentation
Function coverage	<ul style="list-style-type: none">•traces code execution•automated	<ul style="list-style-type: none">•does not ensure generality•misses finer grained branches
Input coverage	<ul style="list-style-type: none">•user-oriented measure•automated	<ul style="list-style-type: none">•does not ensure generality•does not trace code execution

Certain coverage holes are identified by only 1 of the 3.



Non-uniqueness: Granularity

- A **fine-grained** OVTs has many OATs, each with few unique coverage aspects.
- A **coarse-grained** OVTs has few OATs, each with many unique coverage aspects.
- Granularity affects progress measure, test suite size, and diagnostic effectiveness:
 - A **fine-grained** OVTs is better for showing incremental progress and provides easier isolation of coding mistakes and algorithmic weaknesses.
 - A **coarse-grained** OVTs requires fewer tests and less effort to run them (both for initial and sustainable verification).
- A continuous spectrum exists between two extremes.



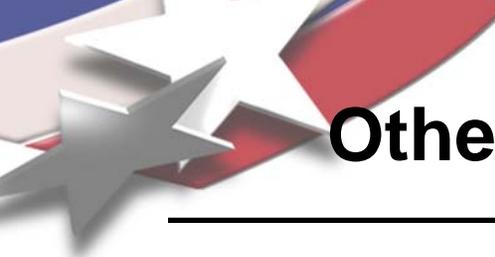
Example Coarse/Fine Decisions

- Initially, all convecting vortex tests (used for temporal integration verification) used the same option for IC enforcement.
- Two new tests were added by copying and modifying existing tests.
- This completed coverage of IC enforcement options, but it also rendered a couple of existing tests redundant.
- These redundant tests were left in the OVTS, and since they have been passed, their presence increases the progress measures.



Example Coarse/Fine Decisions (2)

- An MMS test of the Riemann invariant outflow BC has been executed for fully supersonic conditions, and it passes.
- This test is a less general case of the test of the same BC for mixed subsonic/supersonic conditions, which does not demonstrate asymptotic behavior.
- The fully supersonic case was never added to the OVTS.
- The omission of the fully supersonic case from the OVTS decreases the progress measures.



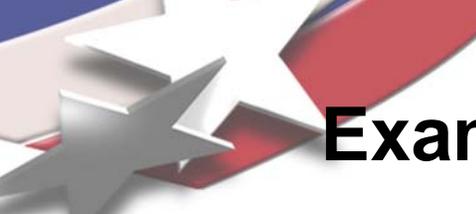
Other OVTS Design Considerations

- In general, exact solutions should be used where they exist, and manufactured solutions should be used to achieve completeness through generality.
- Portions of the space may be covered with unit tests rather than OATs, provided that potential places for coding mistakes and algorithmic weaknesses are not overlooked.
- Some things that meet this criterion may be more conveniently covered by OATs.
- As a side effect, OATs may test things not associated with discretization algorithms, so this ‘bonus’ coverage of the OVTS could shrink the space for other types of testing.

Example OA/Unit Test Decision

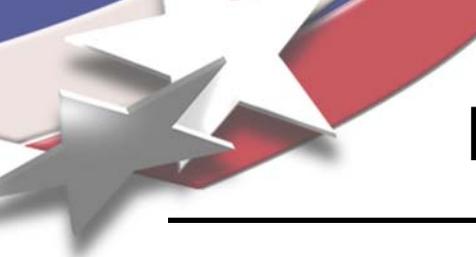
- Viscosity, μ , can be calculated by one of four methods: constant, Sutherland's law, power law, and Keyes' law.
- One function exists with case statements for each option.
- Three options are functions of temperature, T , and also a function of space, since $T = T(x,y,z)$.
- Terms exist in the Navier-Stokes equations which include spatial derivatives of viscosity, for example:

$$\frac{\partial}{\partial x} \left(2\mu \frac{\partial u}{\partial x} \right)$$



Example OAT/Unit Test Decision (2)

- We need the most general functional dependence for μ to test a discretization algorithm for this term.
- Any one of the three options that are functions of T will suffice.
- The other options can be tested with a unit test that calls the function for μ and compares its result with the expected result over a range of T for each option.
- Some OATs within the OVTs use Sutherland's law; other options for μ are tested with unit tests.



Premo OVTS for This Study

- 16 initial OATs
 - 7 convecting vortex tests (unsteady, inviscid, exact solutions) tests used for temporal integration verification
 - 9 steady, MMS tests for spatial order verification of interior equation sets and BC's
- 52 total OATs in current OVTS
 - 9 convecting vortex tests
 - 8 unsteady exact solution tests for limiter options
 - 35 MMS tests
- If done thoroughly, many more tests will be in the final OVTS than in the initial OVTS.



Premo OVTS for This Study (2)

- Old order verification process was
 - create test
 - run test
 - document test
 - move to next test.
- New process involves conceiving a complete OVTS at the beginning, then filling in details and running later.
- The granularity of the OVTS evolved over time to meet various needs.

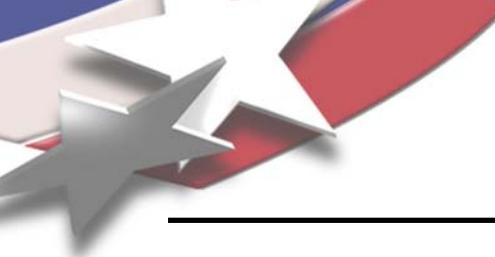


Ways of Calculating Status/Progress

1. Each test is assigned a status level, s , valued 0-5, and the overall measure is

$$P_1 = \sum_{n=1}^N \frac{S_n}{S_{\max}}$$

2. For a weighted measure, each status level is given a different weight.
3. A pass/fail measure is constructed by only giving a non-zero weight to level 5 (*i.e.*, no partial credit).

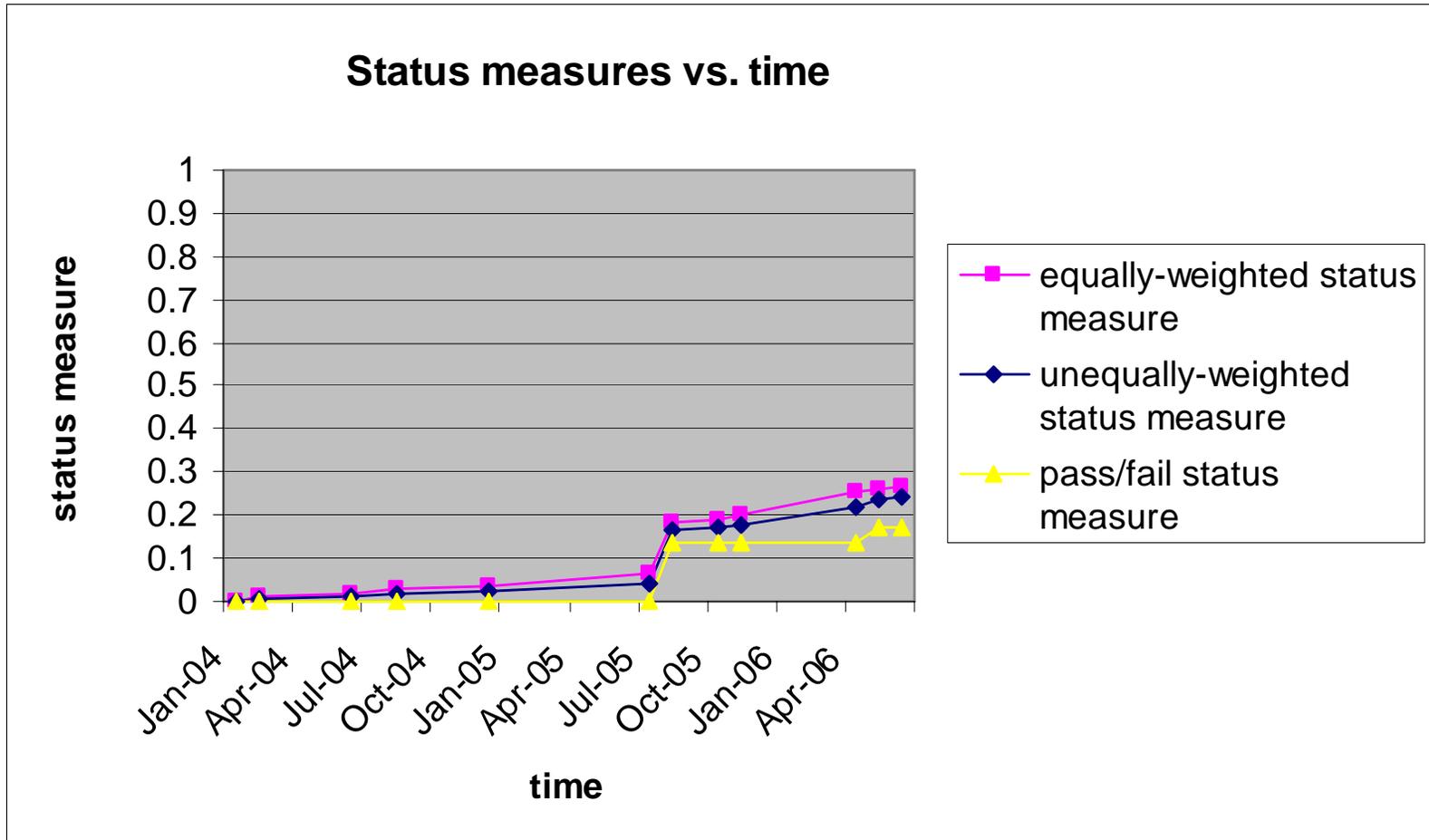


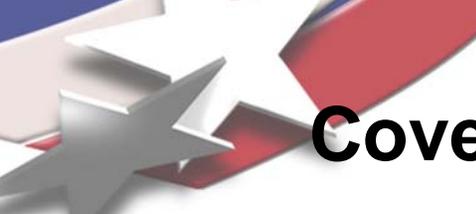
OAT Status Levels

Level	Status	Score
0	incomplete	0 / 20
1	ready	3 / 20
2	numerical solutions exist	5 / 20
3	all solutions asymptotic	10 / 20
4	all orders verified	12 / 20
5	OAT results reproducible	20 / 20



Status over Past 2.5 Years





Coverage-based Progress Measures

- Status and progress measures evaluate progress on a test-by-test basis, and thus are a good indicator of the productivity of verification efforts over time.
- Since significant overlap exists between different tests, the *fraction of tests passed* and *fraction of OV domain verified* are not the same.
- Weighting the progress measure by one of the coverage measures creates a good indicator of the *fraction of OV domain verified*.
- For better or worse, coverage-based measures illustrate the principle of diminishing marginal utility.

Coverage-based Progress Measures (2)

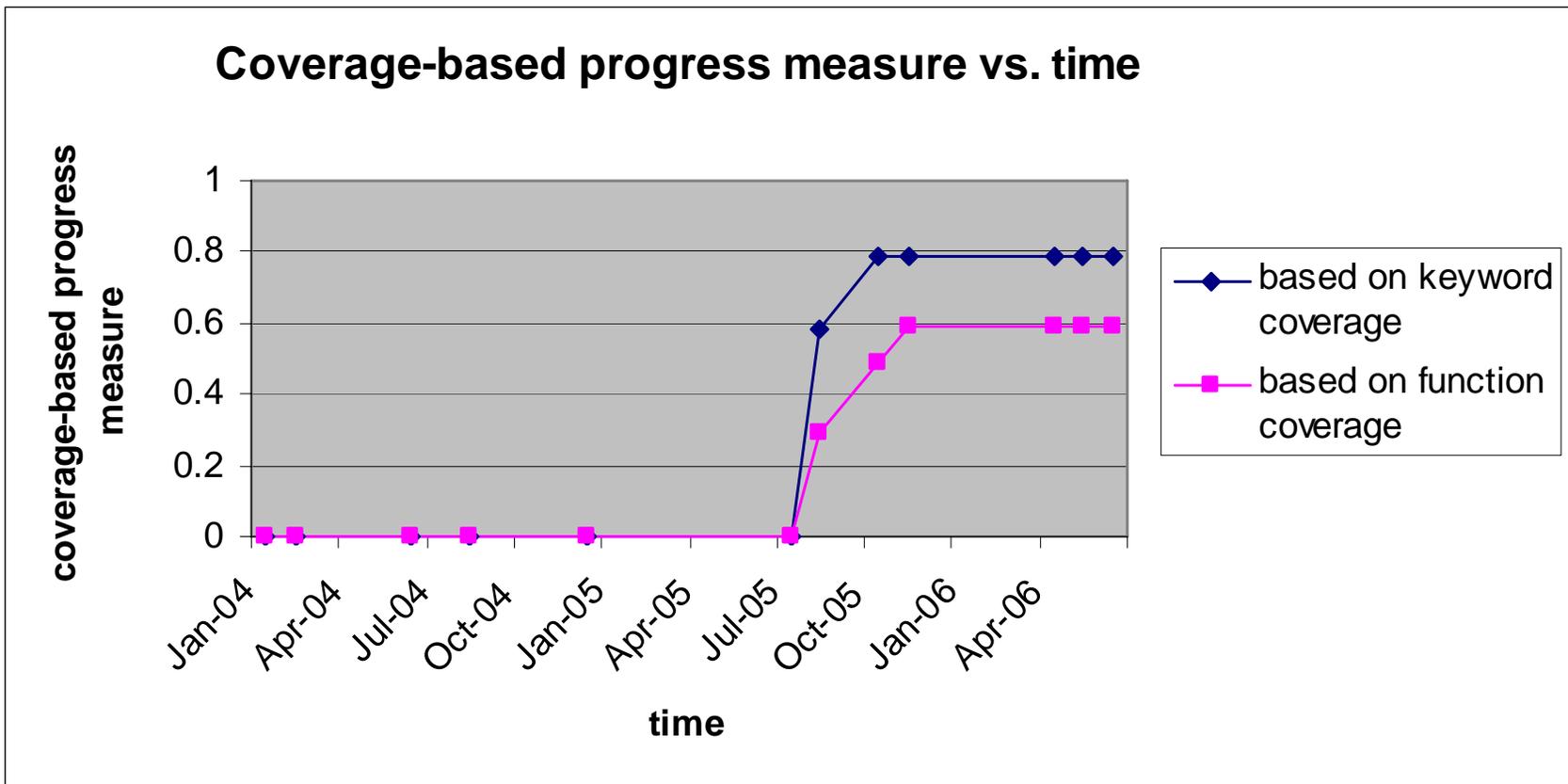
$$P_3 = \frac{\text{functions covered by passed OAT's}}{\text{functions covered by full OVTS}}$$

$$P_4 = \frac{\text{keywords covered by passed OAT's}}{\text{keywords covered by full OVTS}}$$

$$P_5 = \frac{\text{lines covered by passed OAT's}}{\text{lines covered by full OVTS}}$$

These are only useful for a ‘complete’ OVTS, *i.e.*, one that covers all the functions, options, and lines in the OV domain.

Coverage-based Progress Measures (3)





Primary Shortcoming of Coverage-based Measures

Do not take into account the **expert knowledge** measure of completeness, and thus may falsely inflate or deflate the significance of some tests.

example:

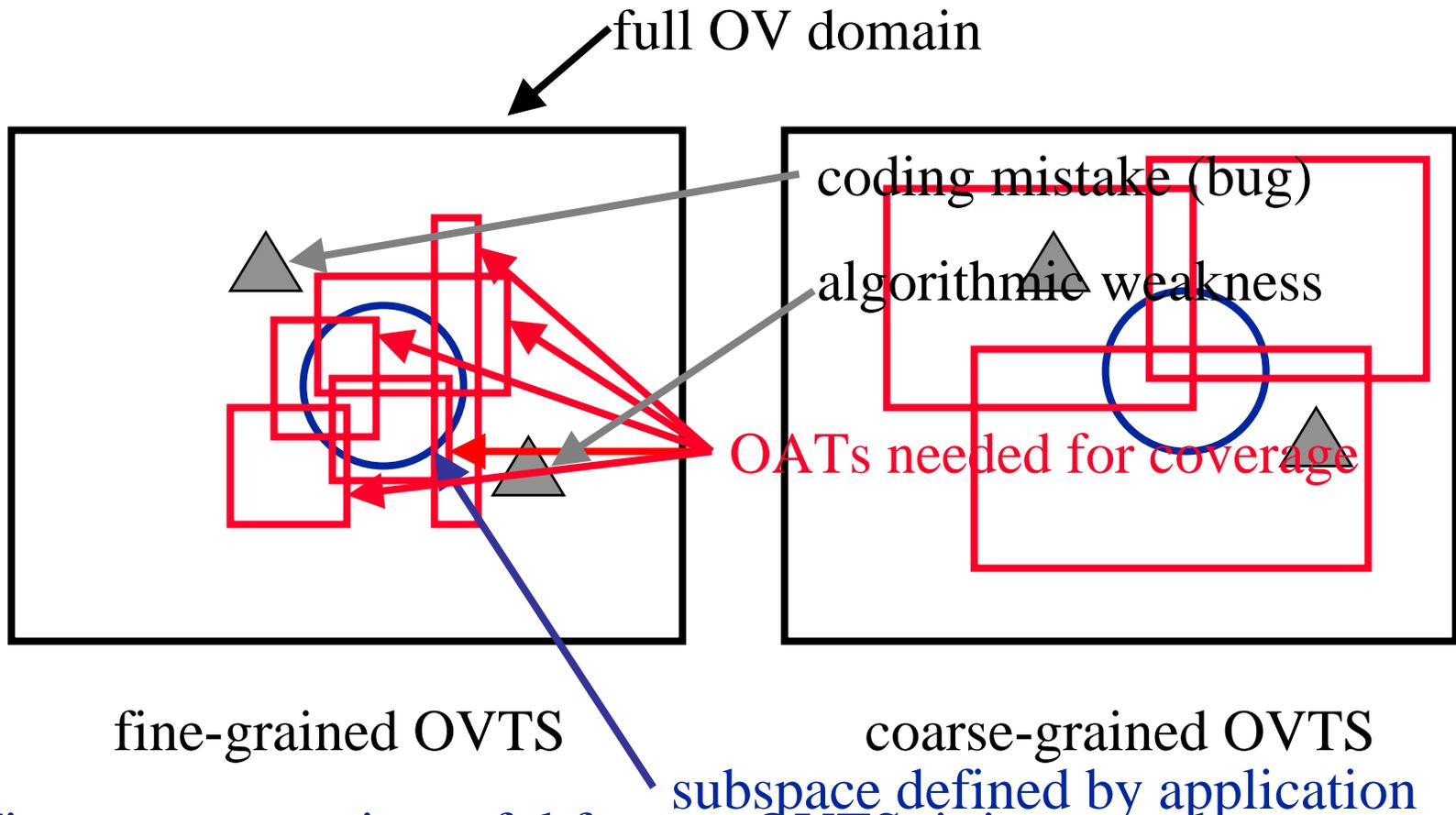
- An exact solution test uses a particular BC, but not in the most general way.
- It is followed by a manufactured solution designed to test the BC generally.
- The automated coverage metrics may miss this distinction, thus showing too large of a jump after the MES test and too small of a jump after the MMS test.



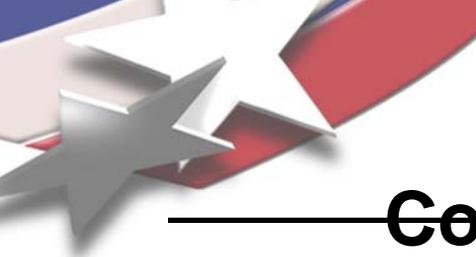
Fitness Measure

- The fitness measure is an indication of a code's verification status relative to a particular application.
- It asks what fraction of the capabilities needed for a specific application has been verified.
- This is difficult to determine accurately for a coarse-grained OVTS, since many of its OATs may exercise features not relevant to the application.
- A high-priority fitness measure calculation might require refinement of the OVTS.
- Whereas the progress measure looks at things from a developer's point-of-view, the fitness measure looks at things from a user's point-of-view.

Impact of Granularity on Fitness Measure



Fitness measure is useful for any OVTS, it just may be more accurate for a fine-grained OVTS.



Progress Measure and Corresponding Fitness Measure

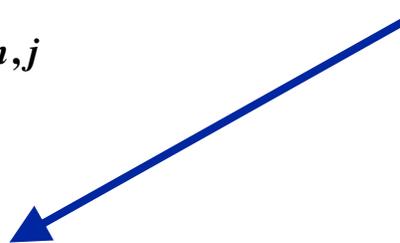
q = observed order of accuracy

p = formal order of accuracy

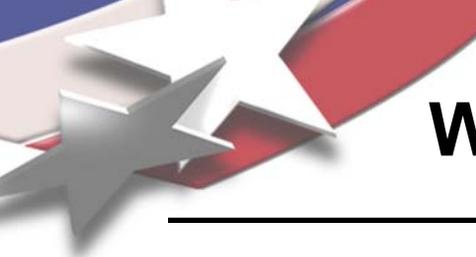
$$r = \begin{cases} \max\left(0, \frac{q}{p}\right) & \text{if } p \text{ is known} \\ \frac{1}{2} & \text{if } p \text{ is not known \& } q > 0 \\ 0 & \text{if } p \text{ is not known and } q < 0 \end{cases}$$

$$P_2 = \sum_{n=1}^N \sum_{j=1}^J r_{n,j}$$

**weight indicating
relevance to application**



$$F = \sum_{n=1}^N \sum_{j=1}^J w_{n,j} r_{n,j}$$



Ways of Determining Weights for Fitness Measure

- Find smallest subset of the OVTs which fully covers the application
 - requires optimization (manual or automated)
 - equal weights for OATs that are members of the subset, zero weights for non-members
 - most similar to status (P_1) and progress (P_2) measures
- Use coverage evaluators to automatically generate weights for all of the OATs within the OVTs
 - measures what fraction of functions, keywords, or lines needed for application are covered by passed OATs
 - more similar to coverage-based progress measures (P_3 , P_4 , & P_5)



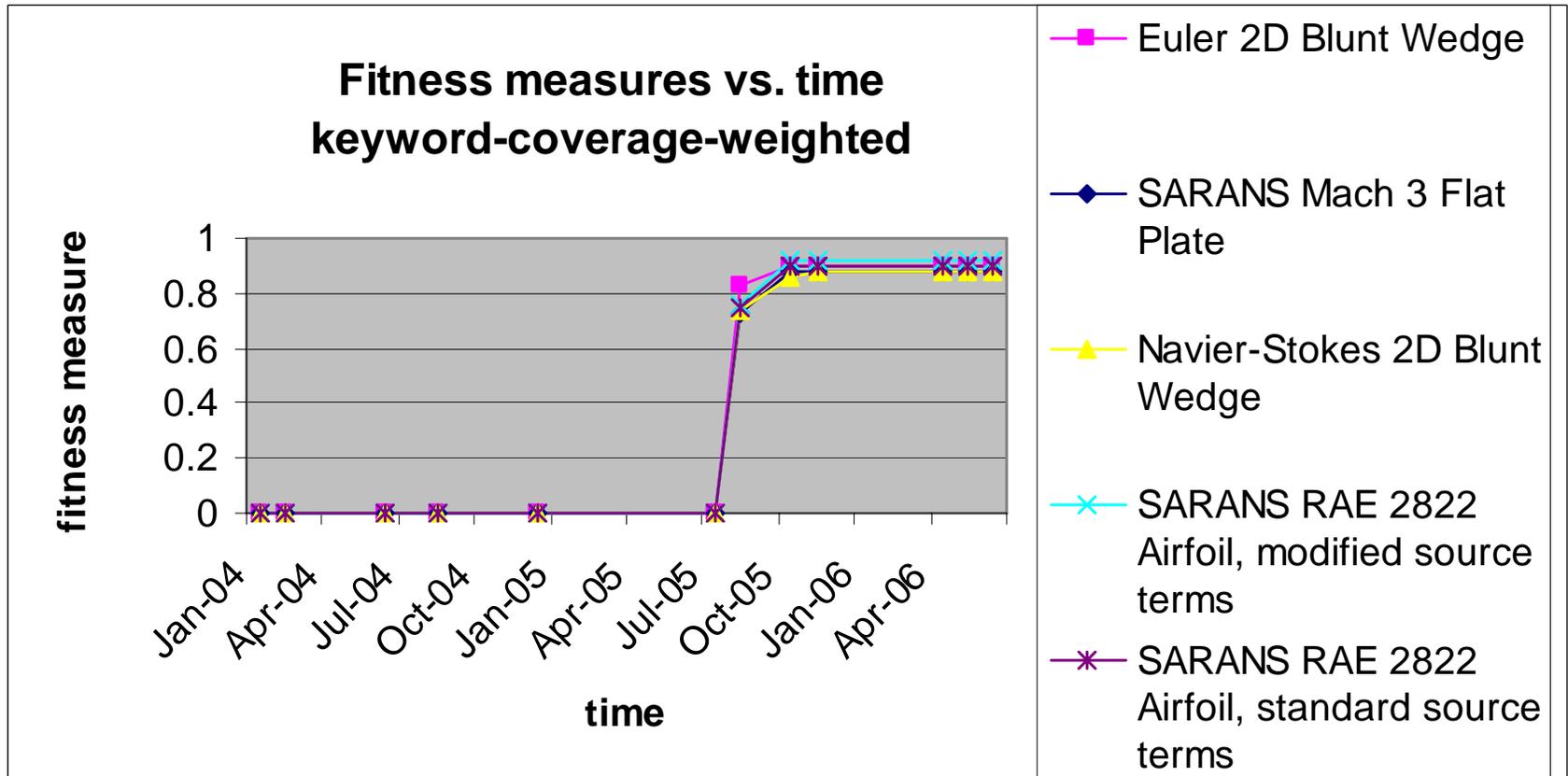
Minimal OVTS Subset Needed for Fitness Measure: Laminar Blunt Wedge

OAT	status level	<i>r</i>
MMS interior Navier-Stokes	5	1
MMS outflow BC, transonic conditions	2	0
MMS outflow BC, stagnation conditions	2	0
MMS farfield BC	0	0
MMS Dirichlet, residual	1	0
MMS adiabatic no-slip, residual	2	0
convecting vortex, uniform IC	1	0
MMS Dirichlet, strong	1	0
MMS adiabatic no-slip, strong	1	0

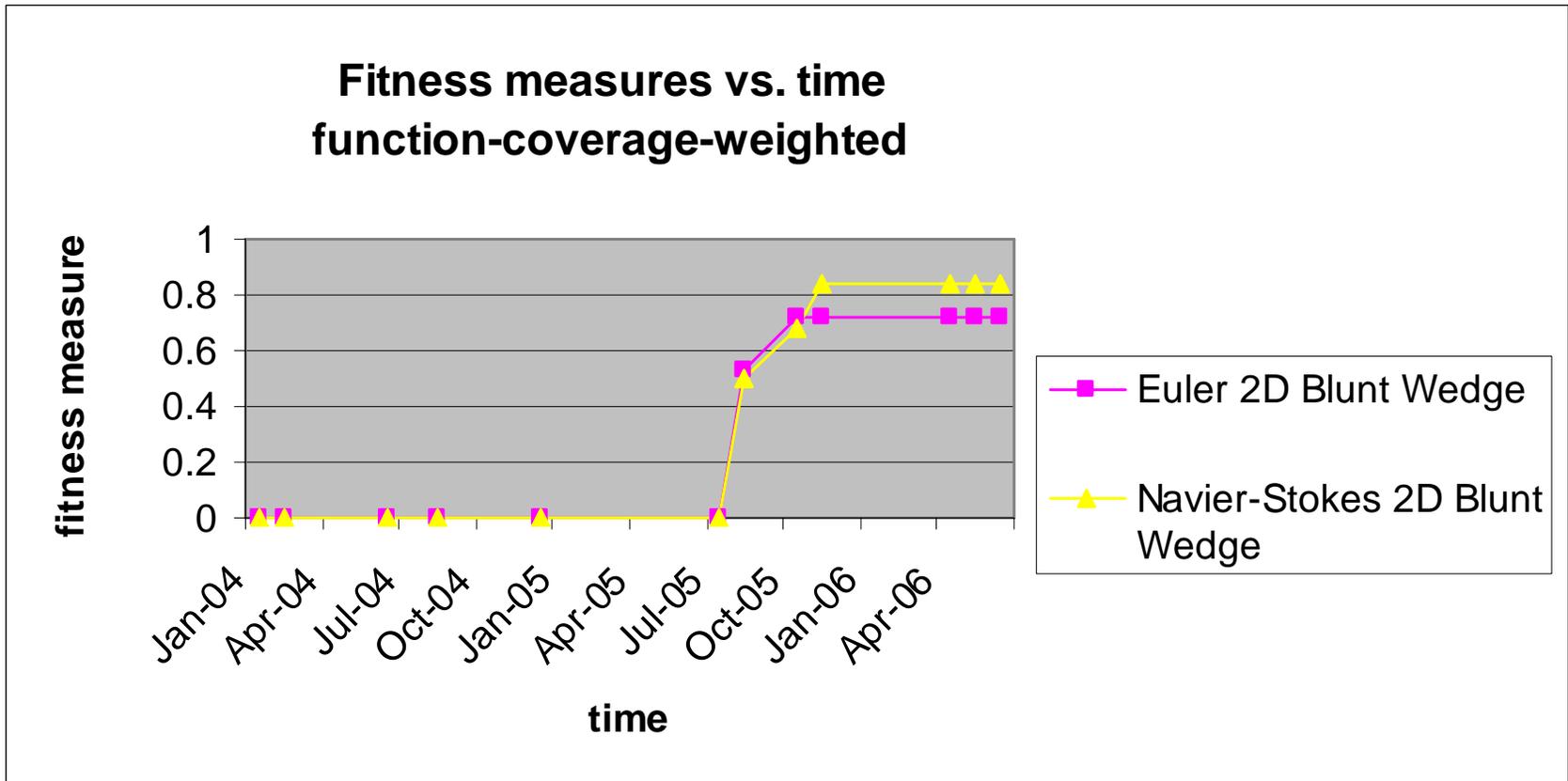
not critical for steady problem

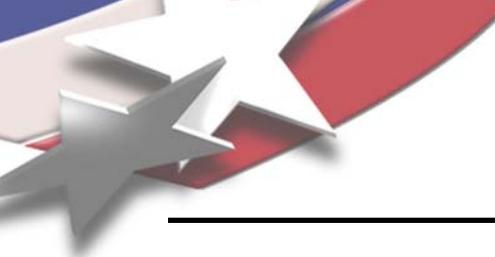


Fitness Measures for Several Example Applications



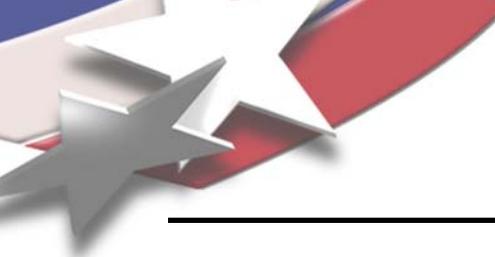
Fitness Measures for Several Example Applications (2)





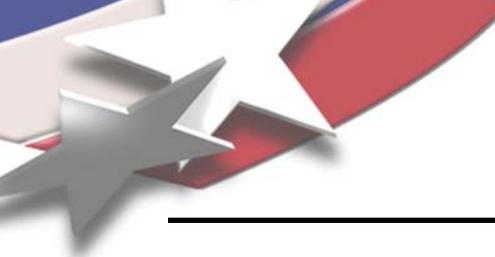
Conclusions

- Evaluating OVTS completeness requires several approaches.
- Automated approaches can be used to compliment expert knowledge.
- The function and keyword coverage methods work reasonably well, but still need some fine tuning.
- Function coverage tends to inflate coverage-based progress measures less than keyword coverage; line coverage may be even more conservative.
- Fitness measures may be even more sensitive to OVTS granularity (and other design decisions) than status or progress measures.



Conclusions (2)

- Sometimes, simply setting up OATs and evaluating them for coverage will uncover coding mistakes.
 - Tests which should have covered certain functions did not, as a result of a missing function call.
 - Tests set up to cover auxiliary variable calculations (for output) threw exceptions in debug mode.
- The granularity of the OATS can be refactored between one code version and another.
 - Tests may be removed (coarser granularity) as tests which duplicate their coverage pass.
 - Tests may be added (finer granularity) to more accurately calculate fitness measures for certain applications.
 - Tests may be added to isolate problems.



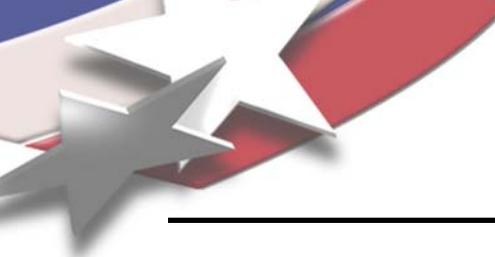
Conclusions (3)

- We should have retained some previously run and passed inviscid OATs in the OVTS in order to
 - increase granularity and get more accurate fitness measures for inviscid problems
 - better represent progress in the earlier stages (late 2004 through early 2005).
- Progress can be measured over time, even if this measure is highly OVTS dependent.
- This study formalized the OVTS and has created an associated database for future investigations.
- The study identified some important practical issues that the theory did not cover.



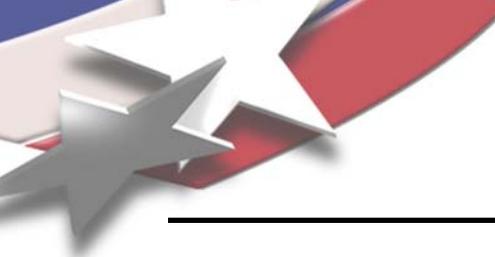
Conclusions (4)

- This study has laid the foundation for measuring progress in OV for Premo and will (or at least should) help guide and prioritize future OV efforts.
- Although we have open issues remaining, this example is sufficient for other code groups to consider emulating.



Open Issues

- Transient code and OVTS versus static code & OVTS.
- OVTS granularity issues need to be examined further.
- Convergence of coverage evaluators to produce consistent mapping between capabilities and pieces of code.
- Development of tools to automate coverage evaluations and progress/fitness measure calculations.
- Generalization of those tools for other codes.



Acknowledgments

The authors would like to thank Jeff Fisher (1541) for his assistance in setting up some tests and modifying Premo and the SNTools to support needed coverage calculations.