Geração Automática de Dados de Teste: *Visão Geral*

Arineiza Cristina Pinheiro

Roteiro

- Introdução
- Técnicas de Busca Meta-heurística
- Técnicas de Teste
  - Teste Estrutural (Caixa-branca)
  - Teste Funcional (Caixa-preta)
  - Teste Caixa-cinza
  - Teste Não-funcional
- Geração de Dados de Teste no Cenário de Sistemas Embarcados
- Considerações Finais

Introdução

- Teste de Software
  - Revelar a presença de defeitos
  - Aumentar a confiabilidade do software
- Escolha criteriosa dos casos de teste
  - Identificar casos de teste
    - Alta probabilidade de encontrar possíveis defeitos
    - Prazo estabelecido
- Independentemente do critério de teste
  - Geração de casos de teste
    - Importante atividade dentro da fase de teste

Introdução

- Caso de Teste
  - Dado de Entrada + Saída Esperada

  - Enumeração Exaustiva
    - Inviável
  - Execução Randômica
    - Não garantem execução criteriosa
    - Trechos dependentes de dados específicos do domínio

Introdução

- Problema Indecidível
  - Complexidade
  - Tamanho dos Programas

- Tecnicas de Busca Meta-heurística
  - *Search-based software teste data generation*
  - Algoritmos de otimização e busca
  - Critério de Teste → Função Objetivo

Técnicas de Busca Meta-heurística

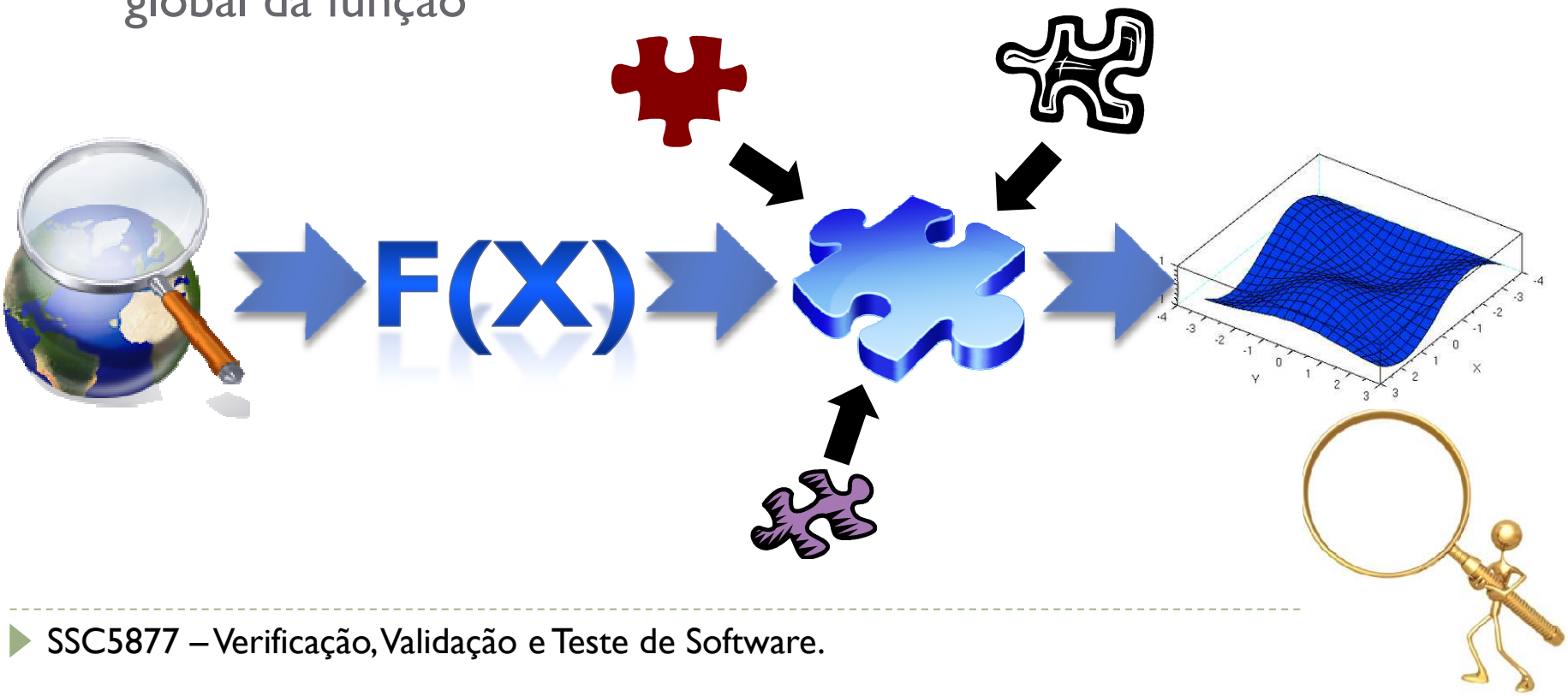
- Técnicas extraídas da área de Inteligência Artificial (IA)
- Problemas de otimização.
- Resolução automática
  - Métodos inteligentes
    - Obtenção de conhecimento durante a sua execução ou previamente
  - Adequação dos dados
    - Em busca do grau de qualidade desejado

Técnicas de Busca Meta-heurística

- Problema de Otimização, dividi-se em (Takaki, 2009):
  - O estado inicial
    - uma possível solução pertencente ao seu domínio que dará origem a busca pela melhor solução
  - As ações possíveis
    - obtenção de conhecimento a fim de adequar a solução
  - Teste de objetivo
    - Determina se a solução encontrada é ótima ou não
  - Função objetivo
    - Avalia quão próximo de uma solução ótima está a solução atual

Técnicas de Busca Meta-heurísitica

- Pesquisa por soluções
  - Em um espaço de busca através de uma função objetivo, que define, matematicamente, o quão próximo a atual solução está da solução ótima, que corresponde a um máximo ou mínimo global da função



Técnicas de Busca Meta-heurística

- Subida de Encosta
  - *Hill-Climbing*
- Têmpera Simulada
  - *Simulated Annealing*
- Algoritmos Evolutivos
  - *Evolutionary Algoritms*
  - Algoritmo Genético

Subida de Encosta

- Analogia da subida progressiva em uma encosta de uma paisagem
- Algoritmo de busca local
- A subida caracteriza o algoritmo iterativo de melhoria
  - Solução inicial aleatória
  - Melhorando-a passo a passo
  - Investigação da vizinhança da solução corrente

Encosta = Vizinhança

Paisagem = Espaço de Pesquisa

Subida de Encosta

- Estratégias de busca:
  - Subida íngreme
    - Toda vizinhança é analisada
    - Elege a melhor solução
  - Subida aleatória
    - Vizinhança explorada randomicamente
    - Substitui a solução corrente pela primeira que oferecer melhor resultado
    - Itera o algoritmo até alcançar a meta estabelecida

Subida de Encosta

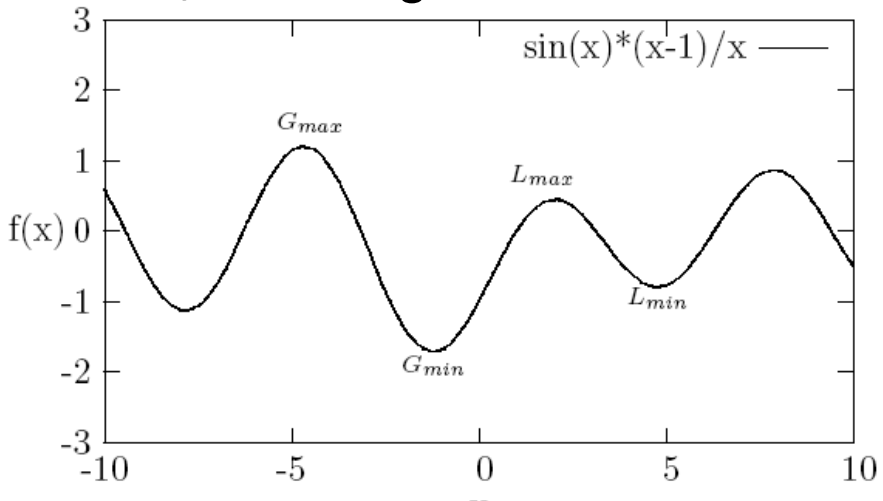
- Algoritmo alto nível
  - Estratégia Subida aleatória

```
Selecione uma solução inicial s ∈ S
Repita
      Selecione s' ∈ N(s) tal que obj(s') > obj(s)
      s ← s'
Até obj(s) ≥ obj(s'), ∀s' ∈ N(s)
```

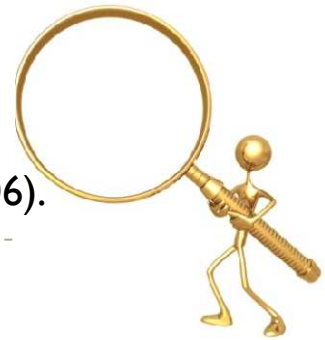
(Adaptado de McMinn, 2004)

Subida de Encosta

- Problemas?!?!?!
  - Algoritmo de busca LOCAL!
  - Explora uma vizinhança do domínio
    - Encontrar uma solução ótima local
  - Objetivo
    - Encontrar uma solução ótima global



Exemplo de uma função objetivo em um espaço de projeto (Abreu, 2006).



Subida de Encosta

- Solução (?!)
  - Reinicalizações
    - Redefini-se a solução inicial
    - Repete o algoritmo
  - Diminui dependência sobre o estado inicial
  - Maior exploração do espaço de pesquisa
  - Aumentam as chances
    - Ótimo Global

Têmpera Simulada

- Analogia com o processo metalurgia de endurecimento de vidros e metais
- Similar ao Algoritmo de Subida de Encosta
- Diferença
  - Recurso que permite retroceder a busca
  - Diminuir a dependência pelo estado inicial

Têmpera Simulada

- Variável de aceitação de solução
  - Probabilidade (p) de a solução investigada substituir a atual
  - $p = e^{-\delta/t}$
    - $\delta = obj(s') - obj(s)$
    - t = controle de temperatura (definido pelo testador)
      - Velocidade de redução de pesquisa

- Aquecimento = Busca pela solução
- Resfriamento = Reinicalização

Têmpera Simulada

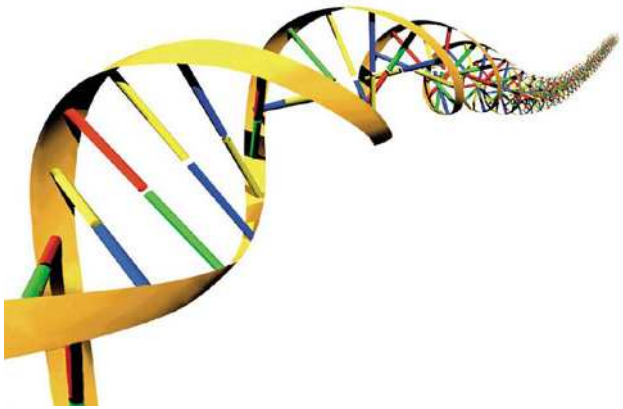
- Algoritmo alto nível

```
Selecione uma solução inicial s ∈ S
Selecione uma temperatura inicial t > 0
Repita
      it ← 0
      Repita
            Selecione s' ∈ N(s) aleatoriamente
            Se obj(s') < obj(s)
                  s ← s'
            Senão
                  Gerar r aleatório sendo 0 ≤ r < 1
                  Se r < e^(-δ/t) então s ← s'
            Fim Se
      Até it = num_soluções
      Decremente t de acordo com cronograma de congelamento
Até alcançar condição de parada
```

(Adaptado de McMinn, 2004)

Algoritmos Evolutivos

- Mais uma analogia
  - Genética e Seleção Natural



- Objetivo
  - Evoluir uma população de indivíduos (candidatos a solução para o problema) através de competição, recombinação e mutação, de forma que a aptidão média da população (qualidade das soluções) seja sistematicamente melhorada dentro do ambiente (problema) em questão (Abreu, 2006).

Algoritmos Evolutivos

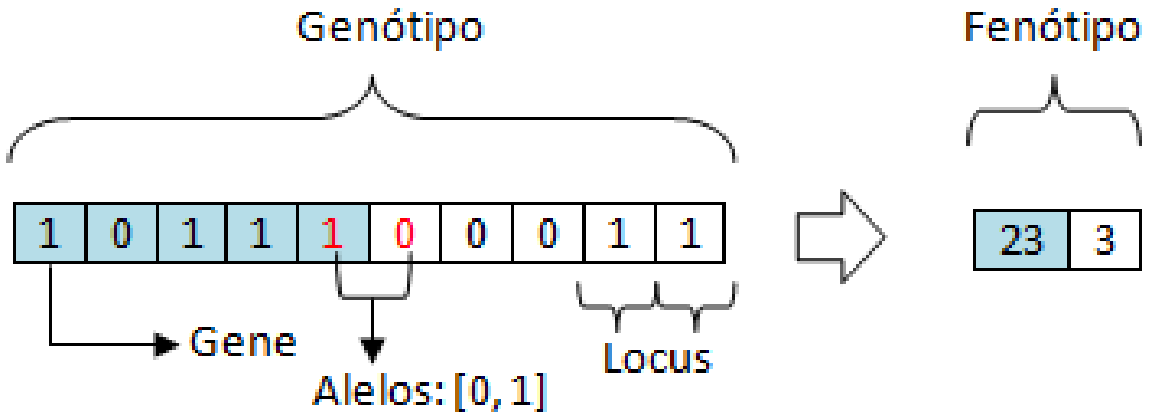
- Analogia entre Solução de Problemas e Evolução Natural

Candidato a Solução	⟺	Cromossomo / Indivíduo
Componentes de solução	⟺	genes
Possíveis valores para cada componente	⟺	alelos
Posição na sequência	⟺	lócus
Estrutura da solução codificada	⟺	genótipo
Estrutura decodificada da solução	⟺	fenótipo

- Algoritmos Genéticos
  - Tipo de AE
    - Baseado no genótipo
  - Outros Tipos
    - Programação Genética e Programação Evolutiva
    - Baseados no fenótipo

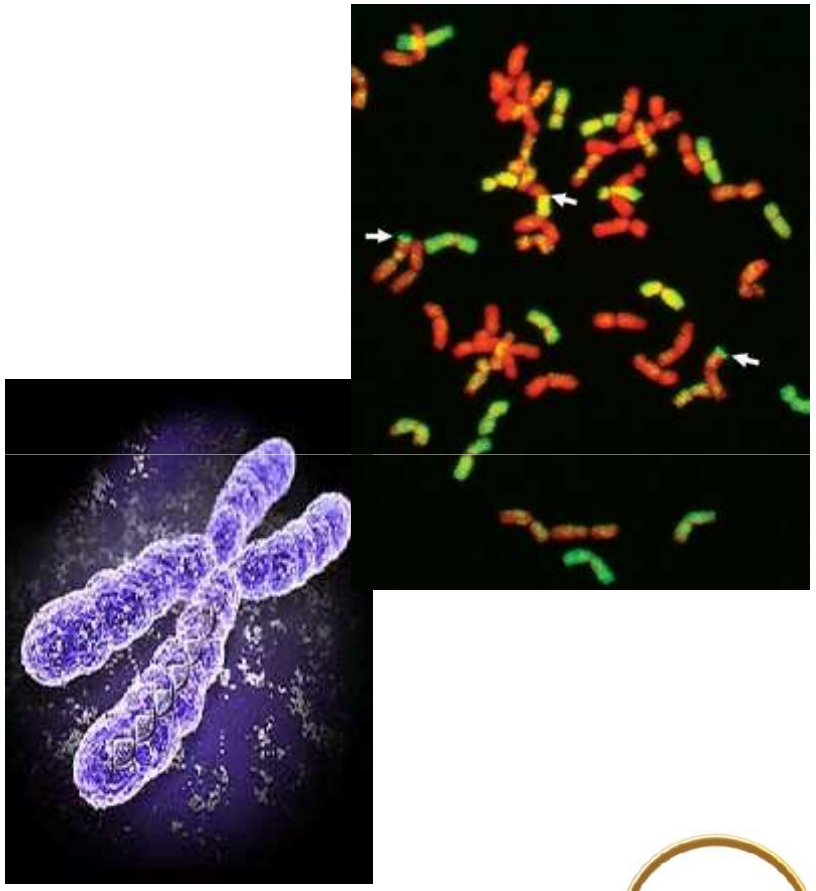
Algoritmo Genético

- Seguindo a analogia
  - Possível Solução → Indivíduo
  - Conjunto de Solução → População
- Indivíduo
  - Valor Real
  - Codificado → String binária



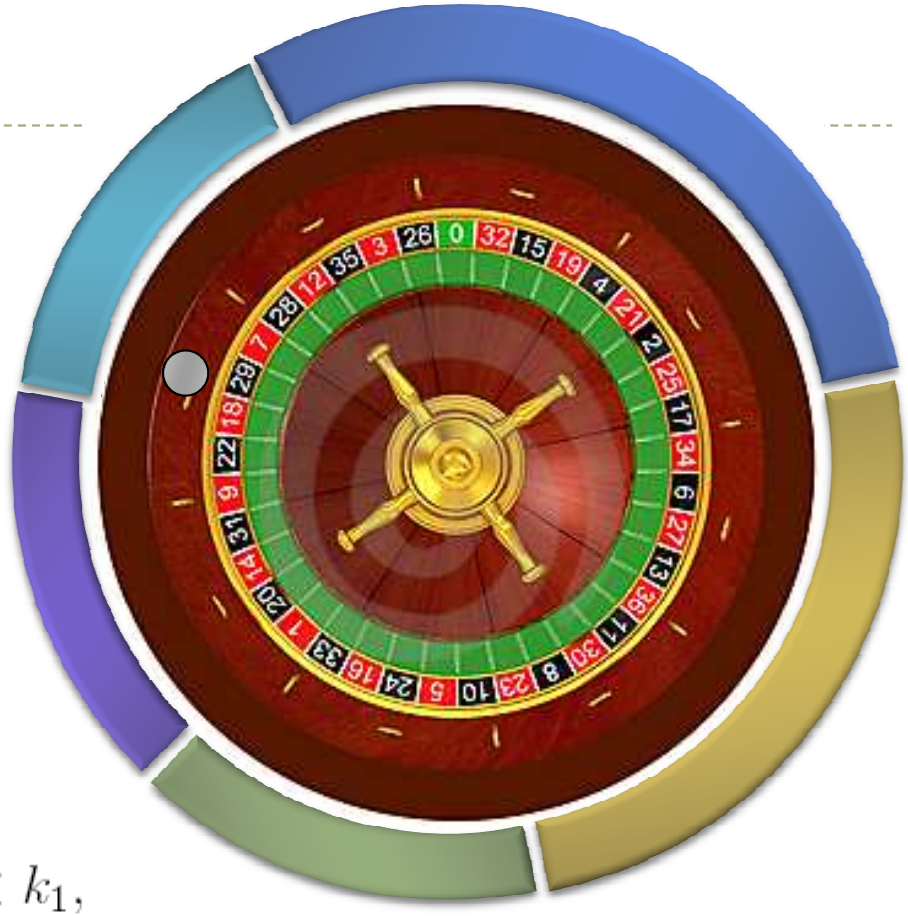
Algoritmo Evolutivo

- Define os métodos
  - Seleção
  - Recombinação
  - Mutação



Algoritmo Genético

- Seleção
  - "Roleta Russa"



$$TFit = \sum_{i=1}^{popsize} fit_i$$

$$p_i = \frac{fit_i}{TFit}$$

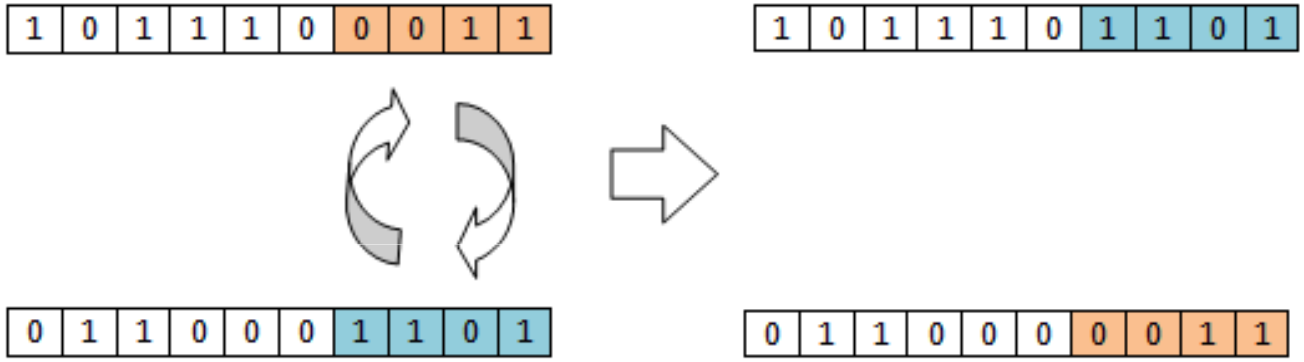
$$k_i = \sum_{j=1}^{i} p_i$$

$$i = \begin{cases} \text{primeiro indivíduo} & \text{se } r < k_1, \\ i\text{-}\acute{e}simo \text{ indivíduo} & \text{onde } 2 \leq i \leq popsize \text{ tal que } k_{i-1} < r < k_i. \end{cases}$$

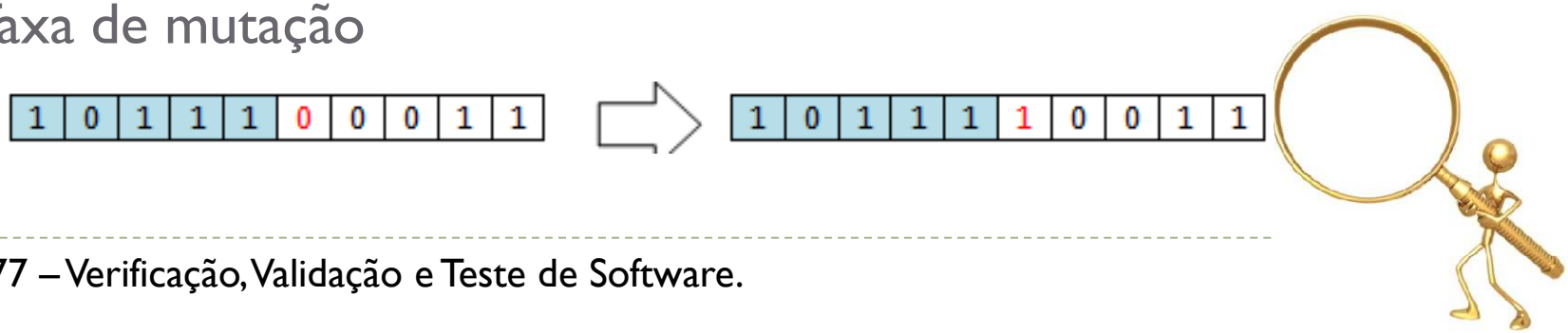
  - r é real pertencente ao domínio [0,1]

Algoritmo Genético

- Recombinação
  - *Crossover*



- Mutação
  - Opcional
  - Taxa de mutação



Algoritmo Genético

- Genótipo
  - String Binária
- Problema
  - Distância entre números
    - 3 e 4 → 011 e 100
  - Teste
    - Limite de variáveis
- Alternativas
  - Código Gray
    - 3 e 4 → 010 e 110
  - Valores reais

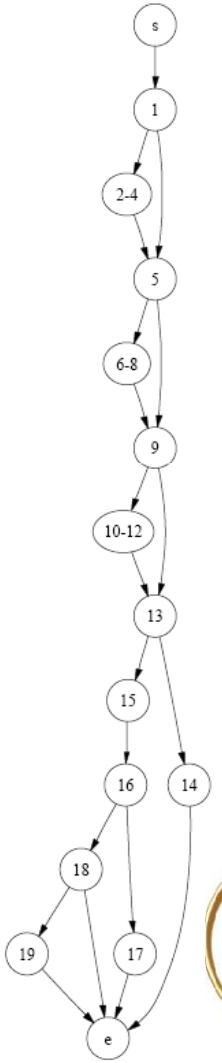
Técnicas de Teste

- Contexto de Geração Automática de Dado de Teste
- Áreas básicas (McMinn, 2004):
  - Cobertura de estruturas específicas do programa
    - Teste estrutural todos-caminhos
  - Execução de características específicas do programa
    - Teste Funcional – Baseado na especificação
  - Abordagem caixa-cinza
    - Simulação de condições de erro
    - Informações da estrutura e da especificação do programa
  - Verificação de características não-funcionais
    - Tempo de execução

Técnica Estrutural

- Conceitos
  - Nós
  - Arcos
  - Caminho
  - Grafo de fluxo de
  - Controle

CFG Node	
s	`int tri_type(int a, int b, int c)`
	`{`
	`  int type;`
1	`  if (a > b)`
2-4	`  {  int t = a; a = b; b = t;  }`
5	`  if (a > c)`
6-8	`  {  int t = a; a = c; c = t;  }`
9	`  if (b > c)`
10-12	`  {  int t = b; b = c; c = t;  }`
13	`  if (a + b <= c)`
	`  {`
14	`  type = NOT_A_TRIANGLE;`
	`  }`
	`  else`
	`  {`
15	`  type = SCALENE;`
16	`  if (a == b && b == c)`
	`  {`
17	`  type = EQUILATERAL;`
	`  }`
18	`  else if (a == b || b == c)`
	`  {`
19	`  type = ISOSCELES;`
	`  }`
	`  }`
e	`  return type;`
	`}`



Programa de classificação de triangulo e correspondente grafo de fluxo de controle (McMinn, 2004)

Teste Estrutural

- Três abordagens principais (Abreu, 2006; McMiin, 2004):
  - Aleatória
  - Execução Simbólica
  - Execução Dinâmica

- Aleatória
  - Valor prático
    - produzir dados próximos aos utilizados operacionalmente
  - Técnica relativamente barata
  - Exige mais do que dados gerados manualmente
  - Não garante cobertura de trechos específicos

Teste Estrutural

- Execução Simbólica
  - Menos utilizada
    - Problemas aumentam a medida que a complexidade do programa também aumenta.
  - Analisa e avalia quais expressões de entrada são necessárias para a execução de um dado caminho
    - Nomes simbólicos são atribuídos às variáveis de entrada
    - Caminhos são avaliados a partir da interpretação das instruções pertinentes, de acordo com estes nomes simbólicos

Teste Estrutural

- Execução Simbólica
  - Computacionalmente cara
    - necessidade de derivação sistemática e manipulação das expressões simbólicas,
  - *Loops*
    - dificultam o monitoramento do caminho desejado
  - Variáveis dos tipos vetor e ponteiro
    - Podem ou não indexar um mesmo espaço de memória já atribuído por outra variável
  - Soluções propostas sem sucesso
    - Aumentam significativamente a complexidade
    - Necessidade de muita memória

Teste Estrutural

- Execução Dinâmica
  - Programa em execução
    - Avalia e melhora os dados à medida que são gerados
  - Define-se um problema de otimização
  - Objetivo
    - Encontrar dados de teste que alcançam o requisito desejado, a partir dos dados coletados durante a execução que se aproximam do requisito, mas que não o satisfazem
  - Altera-se os melhores dados
    - Satisfação do requisito desejado
    - Cobrir uma determinada decisão

Execução Dinâmica

- Primeira solução
  - Redução de código para o trecho de interesse
    - Seleciona-se um caminho
    - Reescreve o código

- Exemplo do Triângulo
  - Caminho
    - < s; 1; 5; 9; 10; 11; 12; 13; 14; e >

```
int tri_type(int a, int b, int c)
{
    int type;
(c1 = (b − a)) >= 0
    int t = a; a = b; b = t;
(c2 = (c − a)) >= 0
(c3 = (b − c)) > 0
(c4 = (c − (a + b))) >= 0
    type = NOT_A_TRIANGLE;
}
```

Execução Dinâmica

- Estabelecidas as restrições
  - Função Objetivo
  - Minimizar para encontrar a solução ótima
- Gera um código novo
  - Instrumentar o código base e executa entrada arbitrária
  - Calcula função objetivo derivada do predicado
    - Distância do ramo

Relational predicate	f	rel
$a > b$	$b - a$	$<$
$a \geq b$	$b - a$	$\leq$
$a < b$	$a - b$	$<$
$a \leq b$	$a - b$	$\leq$
$a = b$	$abs(a - b)$	$=$
$a \neq b$	$-abs(a - b)$	$<$

Funções objetivo para predicados relacionais (McMinn, 2004).

Execução Dinâmica

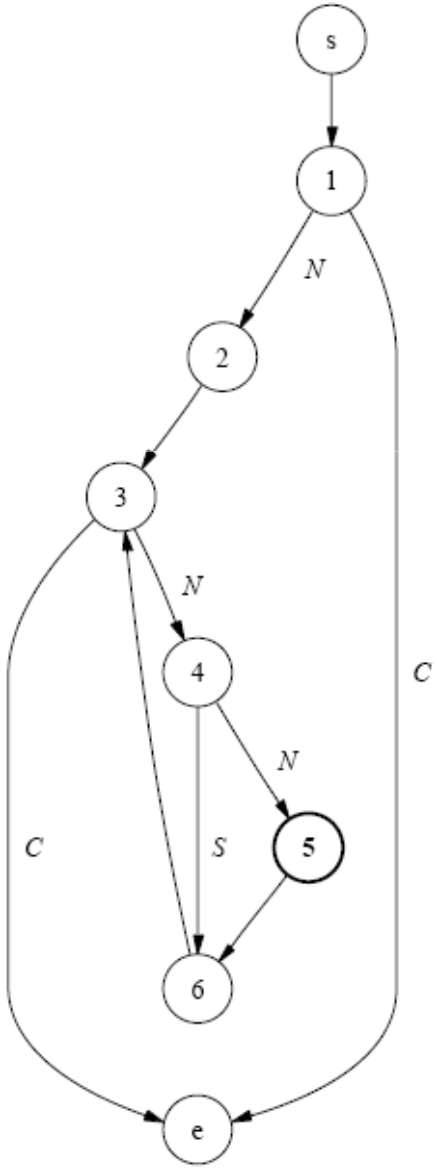
- Porém...
  - Algoritmo de Busca Local
  - Dependente da solução inicial
    - Não obter solução por violar trechos já cobertos
- Avaliar dependências de variáveis
  - Alterar as que influenciam menos o sistema
- Predicados compostos
  - AND
    - Min(f(x) + g(y))
  - OR
    - Min(f(x) || g(y))

Execução Dinâmica

- Técnicas centradas no caminho
- Abordagem orientada a metas
  - Classificação dos ramos no grafo de fluxo de controle
  - Meta: cobrir um nó específico
- Classificação
  - Crítico
    - Não leva por nenhuma ramificação ao nó destino
  - Semi-crítico
    - Está em um *loop* e, inicialmente, não leva ao nó destino
    - Podendo levar em uma próxima iteração
  - Não-essencial
    - Faz parte do caminho que leva ao nó destino

Abordagem Orientada a Metas

CFG Node	
s	`void goal_oriented_example(int a)`
	`{`
1	`    if (a > 0)`
	`    {`
2	`        int b = 10;`
3	`        while (b > 0)`
	`        {`
4	`            if (b == 1)`
	`            {`
5	`                // target`
	`            }`
6	`            b --;`
	`        }`
	`    }`
e	`    return;`
	`}`



Abordagem Orientada a Metas

- Objetivo
  - Evitar arcos semi-críticos quando possível
  - Descartar soluções que executem arcos críticos

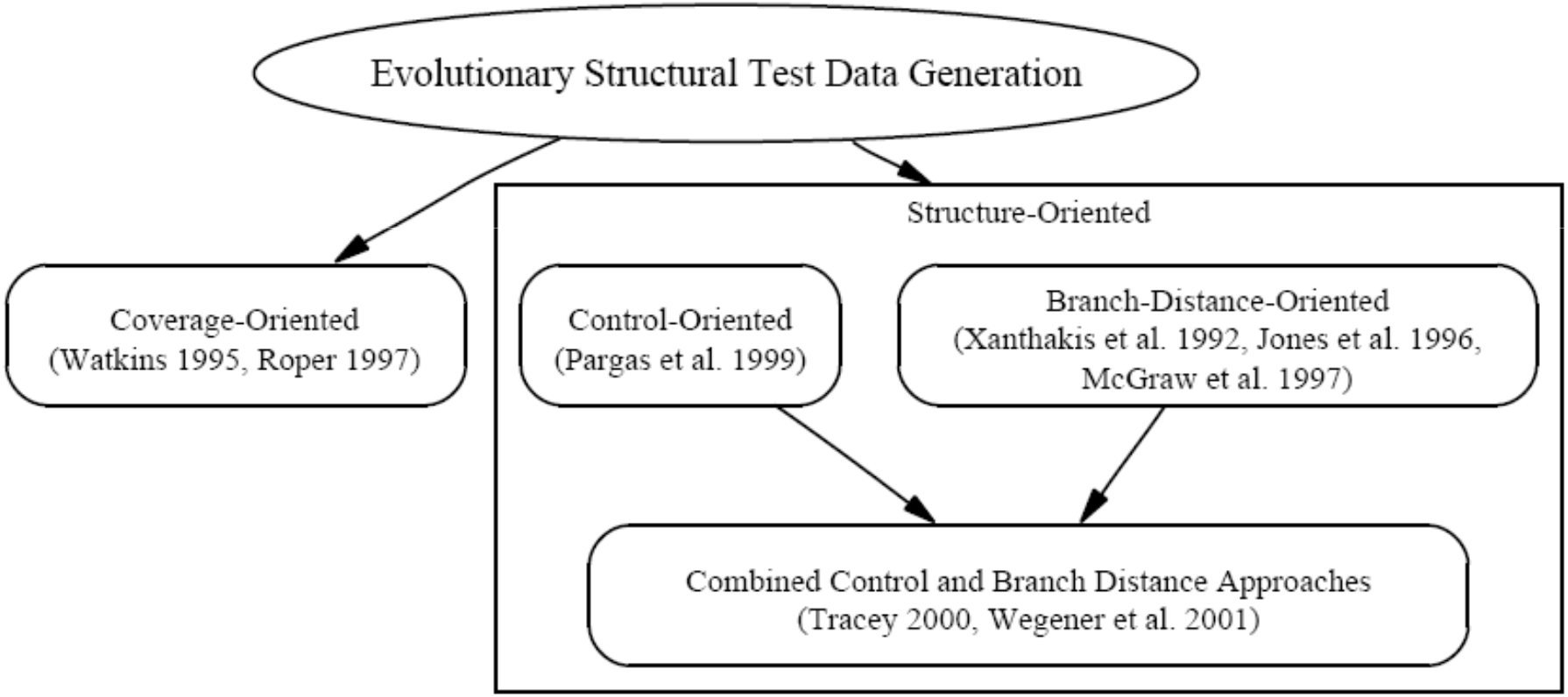
- Porém
  - Busca Local
  - Remoção dos esforços para solucionar um caminho
    - Influência entre variáveis não prevista
    - Prejudicar a execução de nós não-essenciais

Meta-heurísticas

- Têmpera Simulada
  - Retirada de restrição
    - Solução não precisa estar conformidade com um sub-caminho bem sucedido
    - Define-se uma vizinhança para buscar valores
  - Perda de informações
    - Possíveis violações

Meta-heurísiticas

- Algoritmos Genéticos
  - Teste Evolutivo



Abordagem Orientada a Cobertura

- Idéia
  - Indivíduos que executam os caminhos mais longos
    - Recompensados,
  - Indivíduos que executam os caminhos já cobertos
    - Punidos.
- Limitação
  - Dificilmente atinge a cobertura total
  - Sem garantias de execução de trechos específicos
    - Dependência de valores específicos do domínio

Abordagem Orientada a Estrutura

- Explora as informações nos predicados dos ramos

- Linhas
  - Orientada a Distância do Ramo
  - Orientada a Controle
  - Combinação de ambas

Orientada a Distância de Ramo

- Testes randômicos iniciais
- Caminhos não cobertos
  - Seleciona um caminho
  - Tenta cumprir todas as restrições de uma única vez
  - Função objetivo
    - Corresponde à soma de todos os valores de distância dos ramos

Orientada a Distância de Ramo

- Problema 1
  - Seleção do caminho é de responsabilidade do testador
- Solução 1
  - Não escolher um caminho e a função objetivo é simplesmente a distância entre o ramo requerido e o ramo atual
- Problema 2
  - Não há controle sobre os ramos já cobertos anteriormente
    - Violações.
- Solução 2
  - Utilizar os indivíduos que anteriormente garantissem os ramos já cobertos
- Problema 3
  - Ramos dependentes de valores específicos do domínio

Abordagem Orientada a Ramo

- Além disso
  - Aumento das restrições aumenta também as chances de se prender a soluções em ótimos locais
    - Informações orientadas a controle não explicitas precisam ser incluídas na função objetivo

- A Abordagem Orientada a Controle proposta
  - Idéia
    - Fornecer maior *feedback* através da própria função objetivo

Abordagem Orientada a Controle

- Função objetivo
  - considera os nós da ramificação que precisam ser executados
  - Executar a estrutura desejada
- Utilizar um grafo de controle de dependências
  - Valor objetivo de um indivíduo
  - Número Total Dependentes – Dependentes Executados
- Problema
  - Função objetivo com 'platô'
    - Uma área em que a função é constante
    - Minimização sem orientação
    - Busca randômica

Combinado

- Idéia
  - Identifica o grafo de controle de dependências da estrutura
  - Se indivíduo percorre um ramo crítico de um desses nós
  - Cálculo da distância
    - Com base no predicado do ramo requerido
    - E do caminho alternativo que deve ser executado

Teste Estrutural - Conclusão

- Problemas e Desafios
  - Uso de variáveis do tipo enumeração
  - Programas com fluxo de controle desestruturado
    - Estruturas do tipo *case*
  - Sistemas com comportamento baseados em estados
  - Limitação da técnica a problemas numéricos
    - Tratamento para strings
    - Estruturas dinâmicas de dados
      - Árvores e lista,
    - Necessário identificar tamanho requerido e a sua respectiva 'forma'

Teste Estrutural - Conclusão

- Tendências
  - Estudo de teste de programas orientados a objeto
    - Baseados em estados internos
    - Herança e Polimorfismo
    - Apontada por McMinn (2004) e refletida pelo trabalho de Araki (2009)
      - **Um algoritmo evolutivo de geração de dados de teste para satisfazer critérios baseados em código objeto Java**.
      - Test Data Set Generator for OO software (TDSGen/OO)
      - Satisfazer critérios da JaBUTi
  - Teste de programas que utilizam
    - Arquivos
    - *Sockets*

Teste Funcional

- Dados estraídos da especificação
  - Formal
- Especificação Z
  - Declaraçãomatemática do relacionamento entre os módulos que compõe a lógica do programa de modo a satisfazer o seu objetivo funcional

- Exemplo do Triângulo
  - Módulos = rotas
    - Equilátero, Escaleno, Isósceles, Retângulo
    - EntradasVálidas, TriânguloVálido

Teste Funcional

- Objetivo
  - Executar cada uma das rotas
  - Exemplo
    - Equilátero: (x=y) ^ (y=z)

- Similar a técnica estrutural
  - Cálculo de distância
  - Minimizar a função

Teste Funcional

- Teste de Conformidade de Especificação
  - Verificar a conformidade da implementação
  - Em relação a especificação
  - Executa programa com dados de teste gerados
    - Confronta a saída obtida com a saída esperada descrita na especificação.
  - Especificação
    - Descrita matematicamente
    - Função Objetivo = Pré-condição + Pós-condição
    - pré-condição: define o conjunto de entradas válidas
    - pós-condição: define as saídas.

Teste Funcional

- Exemplo

```
int wrapping_counter(int n)
{
    int r;
    if (n >= 10)
        r = 0;
    else
        r = n + 1;

    return r;
}
```

- Pré-condição: $n \geq 0 \wedge n \leq 10$
- Pós-condição: $(n < 10 \rightarrow r = n + 1) \vee (n = 10 \rightarrow r = 0)$
- Função Obj: $obj(n \geq 0) + obj(n \leq 10) +$
$min((obj(n < 10) + obj(r \neq n + 1)), (obj(n = 10) + obj(r \neq 0))$

Teste Funcional

- Tabela Relacional para construção da Função Objetivo

Connective	Objective Function obj
$a \wedge b$	$obj(a) + obj(b)$
$a \vee b$	$min(obj(a), obj(b))$
$a \Rightarrow b$	$obj(\neg a \vee b)$
	$\equiv min(obj(\neg a), obj(b))$
$a \Leftrightarrow b$	$obj((a \Rightarrow b) \wedge (b \Rightarrow a))$
	$\equiv obj((a \wedge b) \vee (\neg a \wedge \neg b))$
	$\equiv min((obj(a) + obj(b)), (obj(\neg a) + obj(\neg b)))$
a xor b	$obj((a \wedge \neg b) \vee (\neg a \wedge b))$
	$\equiv min((obj(a) + obj(\neg b)), (obj(\neg a) + obj(b)))$

(McMinn, 2004)

- Função objetivo
  - Quão perto o dado de teste está de descobrir pontos críticos da especificação
    - Encontrar possíveis falhas

Teste Funcional

- Desafio de pesquisa (McMinn, 2004)
  - Grande variedade de formatos de especificações
    - Gerar dados baseados nas especificações
    - Modelos abstratos de implementação para a sua forma concreta
    - Dificulta a geração automática de dados de teste
  - Tamanho dos espaços de busca
    - Tendem a ser muito grandes
  - Estados internos nos programas
  - Necessidade de uma sequência de inicialização para serem alcançados
    - Necessárias informações adicionais sobre a estrutura de estados do sistema

Teste Caixa-cinza

- Combina informações Estruturais e Funcionais
- Teste de Assertiva
  - Comentários ou código executável

```
/*@ i > 0 and i <= 10 @*/  // assertion
i ++;                      // program statement
```

```
/*@
assert = true;
for (i = 0; i < len-1; i++)
{
    if (a[i] > a[i+1])
        assert = false;
}
@*/
// ... normal program code ...
```

Teste Caixa-cinza

- Transforma as verificações de variáveis em assertivas/afirmações
  - Valor atribuído a uma variável
  - Um estado do programa
  - Verdadeiro ou falso
- Extrair verificações (testes condicionais) do código
- Gerar um código com significado oposto ao original
  - Sua negativa
- Novo arquivo com estrutura equivalente à do código original
  - Inserir *report_violation()*
    - Indica quando ocorre uma violação
    - Caracterizando uma falha do programa original

Teste Caixa-cinza

- Exemplo
  - Assertiva

    $(a < b \wedge \neg(b = c \wedge c = d))$
  - Negativa

    $(a \geq b \vee (b = c \wedge c = d))$
  - Código gerado

```
if (a >= b)
    report_violation();
if (b == c)
    if (c == d)
        report_violation();
```

  - Minimizar função objetivo
    - Formada por sequência de eventos

Teste Caixa-cinza

- Tendências (McMinn, 2004)
  - Incorporação de assertivas arbitrárias dentro de programas
  - Pesquisar dados de teste para verificar a sua violação
    - Conceito muito poderoso a ser explorado

- Aplicações futuras
  - Teste de reuso de componentes
    - Busca por dados que façam com que o componente a ser reutilizado seja chamado nas hipóteses de seu uso seja quebrado
  - Verificação das saídas dos testes estruturais
    - Controlados manualmente

Teste Não-funcional

- Verificação de melhor e pior caso de tempo de execução
  - Sistemas de tempo real
- Estimar o tempo de execução
  - Atividade muito complexa
  - Fatores de hardware e software
- Cálculo do caminho percorrido mais longo ou mais curto
  - Não implica necessariamente no maior ou menos tempo de execução, respectivamente

Teste Funcional

- Teste de tempo de execução baseado em busca
  - Situações que impliquem em tempos extremos de execução
- Função objetivo é simplesmente o tempo de execução em um sistema quando executado com alguma entrada
- Objetivo
  - Maximizar a função objetivo no pior caso
  - Minimizar no melhor caso
- Se um caso de teste encontrado violar as restrições de tempo, a busca pode ser terminada

Teste Não-funcional

- Pesquisas realizadas na indústria (McMinn, 2004)
  - Evidenciam o melhor desempenho de busca realizada por AG
  - Comparado com técnicas randômicas ou com testes elaborados pelos próprios desenvolvedores
- Sugere-se que isto ocorre
  - Dificuldade encontrada para se estimar o tempo real
    - Chamadas de sistema
    - *Linkages*
    - Otimizações do compilador

Teste Não-funcional

- Tendências (McMinn, 2004)
  - Combinar a função objetivo com as utilizadas pela geração de dados de teste estrutural
    - Explorar o comportamento temporal em todos-caminhos
    - Evitar os efeitos da instrumentação
  - Extensão dessas técnicas para software orientado a objetos

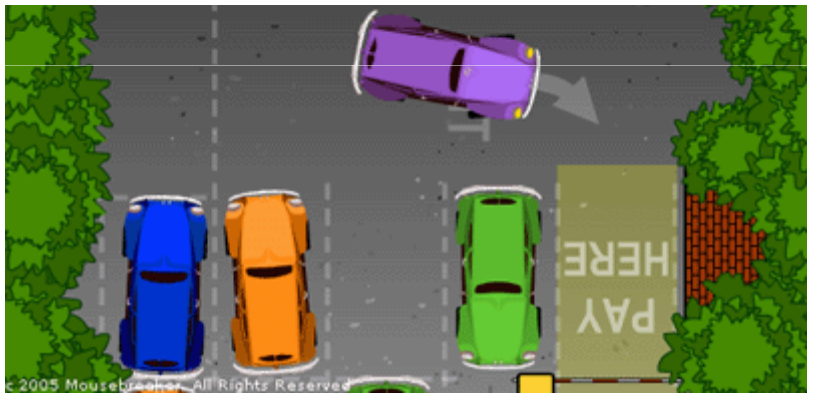
Sistemas Embarcados

- Geração de Dados de Teste no Cenário de Sistemas Embarcados
  - INCT-SEC

- Explorar modelos
  - MEF e MEFE's
  - *Simulink* e *Stateflow*

Sistemas Embarcados

- *Simulink* e *Stateflow*
  - Geração de sinais reais
- Exemplo (McMinn, 2004)
  - Estacionamento automático de carros
  - Vagas longitudinais (90°)
  - Orientação por sensores
    - Áreas de colisão
    - Posição inicial do carro
  - 900 cenários de teste
  - 25 detectaram falhas



Sistemas Embarcados

- Problema deste domínio
  - Geração de sinais reais como dados de entrada
  - Tamanho potencial da entrada
    - Enorme espaço de busca.
- Solução proposta
  - Construir os sinais através de uma série formada a partir de tipos simples de sinais
    - Seno, cosseno e curvas lineares,
  - Espaço de busca um conjunto de parâmetros
    - Definir partes do sinal através de sinais bases
      - Compor através de diversas amplitudes e tamanhos dos tipos base necessários
  - Garante a geração de sinais reais de entrada
  - Mantém o espaço de busca relativamente compacto

Sistemas Embarcados

- MEFE – Máquina de Estados Finitos Estendida
  - Composta por estados, variáveis e transições entre os estados
  - Transição
    - Condições de guarda e operações sobre as variáveis
    - Cumprir condições de guardas → executa transições e ações associadas
- Abordagem poderosa para modelagem
- Amplamente aplicada em muitas áreas
  - Seqüências de teste de sistemas de interesse
- Testes a partir deste modelo
  - Tarefa difícil tarefa devido a dois problemas principais:
    - A viabilidade caminho
    - Geração de dados para o teste de caminhos

Sistemas Embarcados

- Viabilidade caminho é indecidível
- Kalaji, et. al. (2009)
  - Adequação métrica para estimar a probabilidade de um determinado caminho a ser viável
  - A aptidão métrica é baseada na análise de todas as relações entre as transições de um caminho, a fim de estimar a sua viabilidade.
  - Categorizar as transições da máquina de dois tipos: '*affecting*' e '*affected-by*'
    - Uma transição é '*affecting*' em um caminho da transição se tiver uma operação que pode afetar os guardas de uma transição seguinte, os '*affected-by*'

Sistemas Embarcados

- Aborgagem semelhante a Técnica Estrutural
  - Viabilidade de se percorrer um dado caminho
- Examinar caminhos na MEFE para todos os pares
  - (*affecting, affected-by*)
- Atribuir um valor inteiro que estima a sua viabilidade
- Quanto menor o valor da métrica de *fitness* – equivalente a uma função objetivo – mais provável é de que um caminho seja viável

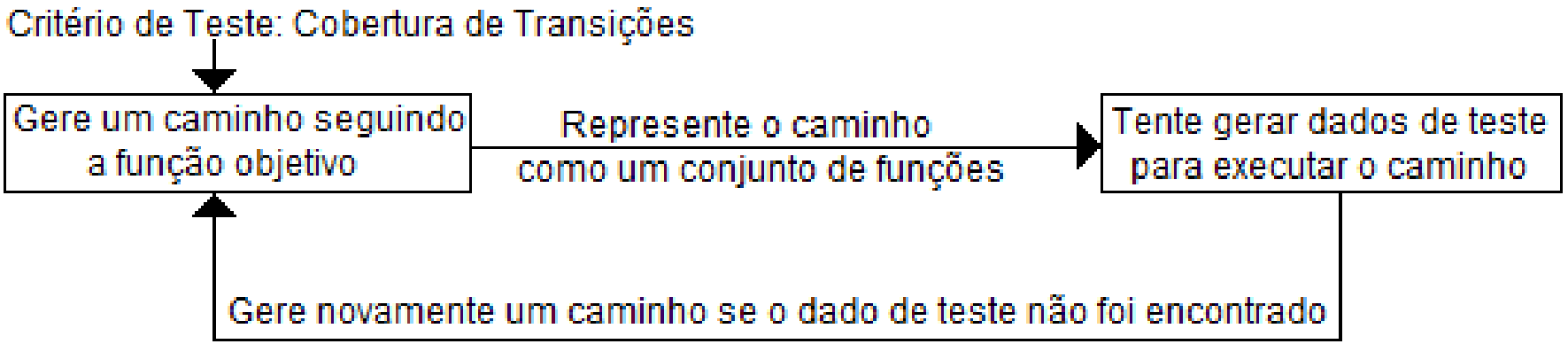
Sistemas Embarcados

- Geração de dados para o teste de caminhos
- MEFE como função
  - Nome da função e os parâmetros de entrada são derivados do nome de transição correspondentes e seus parâmetros de entrada
- Dado de teste para um caminho
  - Conjunto de fatores de produção a ser aplicado a um conjunto de funções que são chamadas sequencialmente
- Função objetivo
  - Orienta a busca de um conjunto adequado de dados de teste

Sistemas Embarcados

- Função objetivo
  - Função de distância + Função de nível de aproximação.
- Função distância
  - Assume o valor zero quando a transição correspondente é acionada
  - Caso contrário, a uma função de distância traduz o quão perto uma dada entrada está de executar o caminho desejado
- Função de nível de aproximação
  - Número de funções críticas (transições) que desviam do alvo
  - Cada nó crítico corresponde a uma função guarda no caminho executado
- Minimizar a função objetivo seguindo o framework proposto por Kalaji et. al. (2009)

Sistemas Embarcados



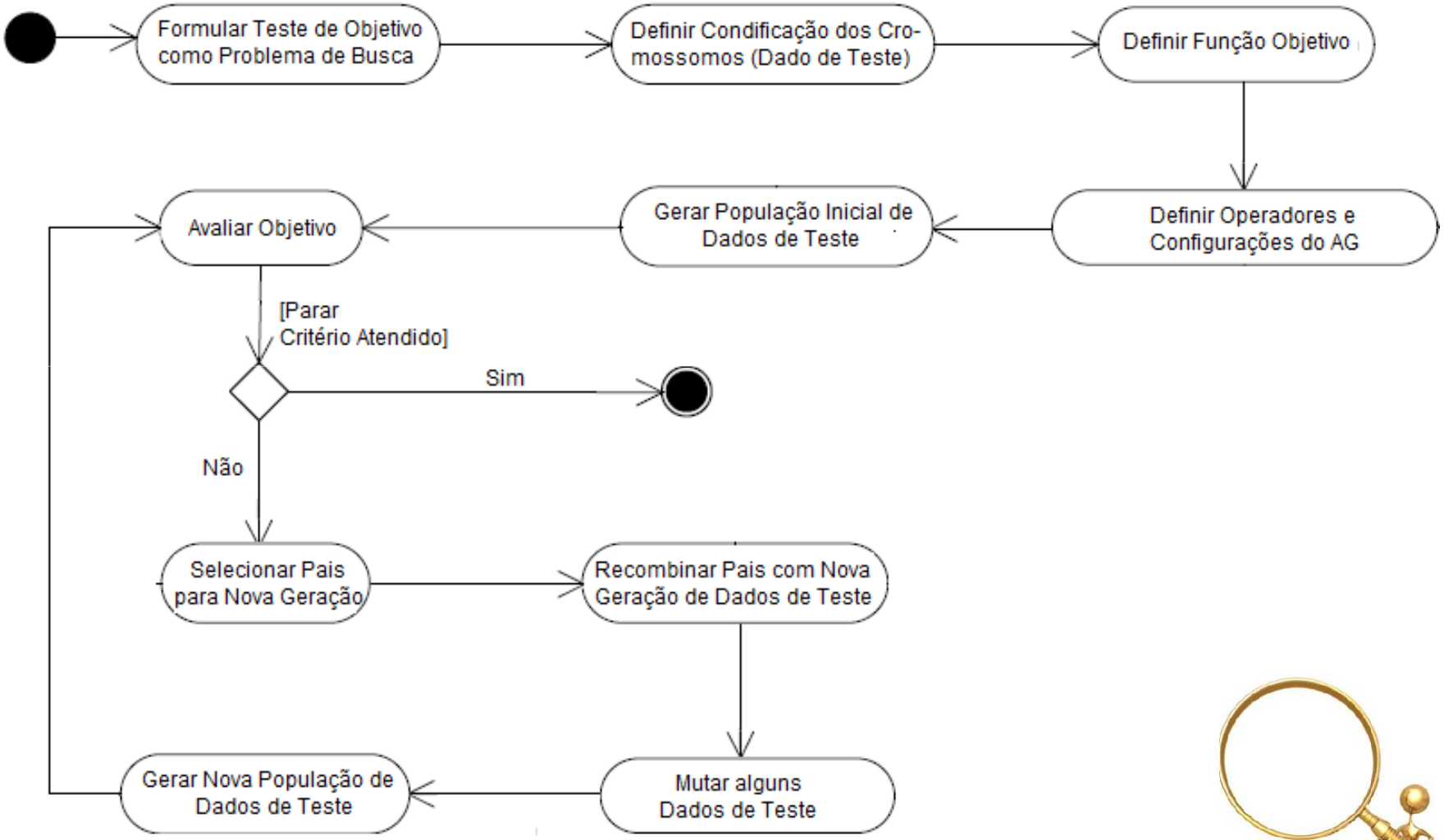
(Adaptado de Kalaji et. al., 2009)

- 1
  - A função objetivo é usada por uma abordagem evolutiva para gerar um caminho de transição que possa ser viável
- 2
  - Utiiliza novamente uma abordagem evolutiva para tentar executar o caminho proposto pelo cálculo da função objetivo
- Se a segunda fase não for bem sucedida ocorre a iteração

Considerações Finais

- Desafios em todas as áreas
- AG são recorrentes
  - Ali et. al. (2008)
    - Diagrama que sumariza a geração de dados de teste por AG

Considerações Finais



Considerações Finais

- Apesar dos desafios na área de geração automática
  - Muitas evoluções nas técnicas estudadas
  - Passo importante para um contínuo estímulo na pesquisa
- Atividade
  - Importante
  - Cara
  - E, de certa forma, ineficiente
- No cenário de Teste de Software.

- 3rd International Workshop on Search-Based Software Testing - http://www.cse.unl.edu/~myra/sbst2010/

Referências

- ABREU, B. T.. **Uma abordagem evolutiva para a geração automática de dados de teste**. 2006. 131 f. Dissertação (Mestrado em Ciências da Computação) – Instituto de Computação, Universidade Estadual de Campinas, Campinas, 2006.
- ALI, S.; BRIAND, L. C.; HEMMATI, H.; PANESAR-WALAWEGE, R. K. A systematic review of the application and empirical investigation of search-based test-case generation. **IEEE Transactions on Software Engineering**. p. 1-22, 2010
- ARAKI, L. Y. **Um algoritmo evolutivo de geração de dados de teste para satisfazer critérios baseados em código objeto Java**. 2009. 102 f. Dissertação (Mestrado em Informática) – Setor de Ciências Exatas, Universidade Federal do Paraná, Curitiba, 2009.
- KALAJI, A.; HIERONS, R. M.; SWIFT, S. A search-based approach for automatic test generation from extended finite state machine (EFSM). **Testing: Academic and Industrial Conference - Practice and Research Techniques** (TAIC PART '09), Windsor, 2009. p. 131-132.
- MALDONADO, J. C.; JINO M.; DELAMARO, M. E. Conceitos Básicos. In: MALDONADO, J. C.; JINO M.; DELAMARO, M. E. **Introdução ao Teste de Software**. São Paulo: Elsevier Editora Ltda., 2007, v. 1, p. 48-76.
- TAKAKI, M. **Busca meta-heurística para resolução de CSP em teste de software**. 2009. 118 f. Dissertação (Mestrado em Ciências da Computação) – Centro de Informática, Universidade Federal de Pernambuco, Recife, 2009.
- MCMINN, P. Search-based software teste data generation: A survey. **Software Testing, Verification and Reliability**, v. 14, n. 2, p. 105-156, p. 105-156, jun. 2004.

Geração Automática de Dados de Teste: *Visão Geral*

Arineiza Cristina Pinheiro