Algoritmo genético para resolução do problema do caixeiro viajante

Wederson Adriano Lourenço da Silva[1]

[1]Unipac – Universidade Presidente Antônio Carlos
Rodovia Deputado Zezinho Bonifácio, Km 12 – Colônia Rodrigo Silva – 36.200-000 – Barbacena – MG – Brasil

wederson_adriano@hotmail.com

Abstract. *This article presents the study and development of an application for solving the Traveling Salesman Problem (TSP) by means of genetic algorithm (GA), which aims to make appropriate parameterization and uses of new experiments. The results show that the population increase in the number of generations and provide for significant improvements which, in turn, cause the increase of time spent. To circumvent this problem are discussed techniques for parallelization of the algorithm.*

Resumo. *Este artigo apresenta o estudo e desenvolvimento de uma aplicação para resolução do Problema do Caixeiro Viajante (PCV) por meio de algoritmo genético (AG), os quais, visa tornar oportuno a parametrização e utilizações de novos experimentos. Os resultados mostram que o aumento da população e do número de gerações proporcionam melhoras significativas que, em contrapartida, ocasionam no aumento do tempo gasto. A fim de contornar este problema, são abordadas técnicas para a paralelização do algoritmo.*

1. Introdução

O Problema do Caixeiro Viajante (PCV) descreve a complexidade de encontrar o ciclo hamiltoniano de menor custo. O PCV representa um problema de grafos composto por um conjunto de n vértices (cidades) e um custo entre cada par de cidades, cujo, o objetivo, é encontrar dentre as (n-1)! possibilidades, aquela que passa por todas as cidades e de menor custo. Por se tratar de um problema de otimização combinatória, atrai pesquisadores de diferentes áreas, como: matemática, engenharia e ciência da computação (GUEDES, 2005).

Sua descrição, embora simples, revela um problema de classe NP-difícil (GUEDES, 2005), o que o torna intratável na obtenção de soluções exatas.

O problema tem uma aplicabilidade em diversas áreas tais como: indústrias (minimizar custo por meio da melhor rota utilizada por equipamentos no processo de montagem), empresas (transportadoras, entrega e coleta de cargas), automobilística (redução de custo de viagem), transporte de passageiros, roteirização de serviços de reparos ou serviços públicos (como coleta de lixo, entrega postal), sequenciamento de genoma.

Diante de algumas de suas várias aplicações, pode-se ter uma visão dos grandes benefícios que uma solução ótima pode trazer à sociedade. O grande desafio compreende em encontrá-la, visto ser de complexidade fatorial e, intratável a solução exata para problemas de grande porte.

Busca-se abordagens aproximativas (heurísticas/metaheurísticas) e exatas uma vez que não pode ser tratado por algoritmos polinomiais.

Avanços na área foram obtidos por meio de trabalhos com abordagem:

- Exata: (DANTZIG et al., 1954), (GRÖTSCHEL, 1980), (CROWDER e PADBERG, 1980), (PADBERG e RINALDI 1987, 1991), (GRÖTSCHEL e HOLLAND, 1991), (APPLEGATE et al., 1995);
- Heurísticas: (LIN e KERNIGHAN, 1973), (CHRISTOFIDES, 1976), (ROSENKRANTZ et al., 1977) e(GOLDEN et al., 1980);
- Metaheurísticas: (REEVES, C. R., 1993), (JOHNSON, D.S. E MCGEOCH, 1997), (GLOVER, 1999), (HANSEN e RIBEIRO, 2001), (GLOVER e KOCHENBERGER, 2003) e (IBARAKI, T. Et al., 2005).

Em 20 de abril de 2001, David Applegate, Robert bixby, Vasek Chvátal e William Cook anunciou a solução de um PCV contendo 15.112 cidades da Alemanha (problema d15112 do TSPLIB - "Traveling Salesman Problem Library") (TSPLIB, 2010), considerado um dos maiores exemplos da TSPLIB, contendo duração de 1.573.084 unidades, correspondendo a cerca 66.000 km. O cálculo foi realizado em uma rede de 110 processadores das universidades de Rice e Princeton, em que foram gastos 22,6 anos em tempo de computação, escalado para processador Alpha EV6 Compaq a 500 Mhz. (TSP, 2010).

A cada ano novos desafios contendo um maior número de vértices são propostos e postados na TSPLIB contendo melhor solução (outro exemplo seria o Lisa100K.tsp-mona contendo 100.000 cidades). Outro grande ponto de referência que pode ser citado é (DIMACS, 2010), entre os quais, pode-se encontrar desafios e comparativos entre variadas abordagens.

Com os exemplos citados, tem-se uma concepção abrangente do problema que, apesar de o hardware avançar cada vez mais, com novas tecnologias, com poder de processamento maiores e, de todos os avanços, tende a chegar a um limite que, a otimização de código, na busca de soluções exatas ou aproximativas, podem, de alguma forma ter melhoras significativas.

O foco deste trabalho é avaliar e estudar a aplicação de AG (Algoritmo Genético) para resolução do PCV em diferentes metologias aplicadas (hibridas, heurísticas, metaheurísticas), como: infecção viral, colônia de formigas, entre outras. A partir do estudo e avaliação da literatura existente, implementar algoritmo genético para resolução do Problema do Caixeiro Viajante e realizar comparações por meio da TSPLIB e outros algoritmos.

No próximo item serão descritos o PCV e algumas das metodologias aplicadas

na sua resolução. No item 3 serão abordados: algoritmos genéticos, implementação, argumentos de otimização da JVM – *Java Virtual Machine*, a estrutura do algoritmo implementado neste trabalho e o paralelismo através de bibliotecas disponibilizadas pelo java. No item 4 serão apresentados experimentos e resultados obtidos, bem como comparativos de resultados constantes na literatura. Conclusão e propostas futuras serão vistos no item 5.

2. Problema do Caixeiro Viajante

Nesta seção serão abordadas algumas das principais metodologias aplicadas na resolução do PCV, como:

- Métodos exatos
- Métodos heurísticos
- Algoritmos genéticos
- Algoritmos meméticos
- Algoritmos transgenéticos
- Otimização por nuvem de partículas

Os métodos exatos compreendem a resolução do problema de forma natural, porém, inviável para problemas de grande porte, uma vez que a solução é obtida pela combinação de todas as possibilidades. Tendo em vista tal complexidade, foram desenvolvidas outras formas de resolução do problema que, baseadas em programação inteira, visam a garantia de uma solução ótima em tempo finito e ou provar que uma solução viável não existe (PRESTES, 2006). Alguns dos métodos que utilizam essa técnica: Branch & Bound, Branch & Cut, Branch & Price, Relaxação Lagrangeana e Programação Dinâmica.

Segundo Dumitrescu e Stützle, citado por (GUEDES, 2005), as vantagens e desvantagens dos métodos exatos são:

1. Vantagens: provar que soluções ótimas podem ser obtidas se o algoritmo tiver sucesso na execução, obtenção de limites inferior e superior em relação a solução ótima e, desconsideração do espaço de busca em que a solução ótima não pode ser encontrada.
2. Desvantagens: limitação pelo alto custo computacional e consumo de memória.

Métodos heurísticos não garantem uma solução ótima, mas busca por soluções aproximadas e ou tempo de execução aceitáveis sem possuir limites formais de qualidade (falta de precisão). Apesar da falta de precisão, tais métodos, de certa forma, se comportam como a inteligência humana, que por meio de sucessivas aproximações direcionadas a um ponto ótimo, obtém soluções viáveis. São exemplos de algoritmos heurísticos: Busca Tabu, Simulated Annealing (SA), Greedy Randomized Adaptive Search Procedures - GRASP, Algoritmo Lin- Kernighan (LK), Iterated Lin-Kernighan, LK-ABCC, LK-H, LK-NYYY. (PRESTES, 2006)

Algoritmos Genéticos (AG) tratam de uma implementação segundo a teoria da seleção natural de Darwin, onde, indivíduos mais aptos sobrevivem. Utiliza operadores de reprodução, cruzamento e mutação. São geradas novas populações a partir destas operações que, por meio da combinação de melhores soluções correntes, gradativamente são aprimoradas. Por ser o foco deste trabalho, terá uma melhor abordagem na seção 3. Referências para exemplos aplicados ao PCV podem ser encontrados em (PRESTES, 2006).

Algoritmos Meméticos (AM) são considerados extensões de AG's, diferenciando-se apenas pela questão de evolução, que neste caso, simula uma evolução cultural e não biológica. Dawkins, citado por (PRESTES, 2006), define um meme (de memes – conceito que originou o nome) como informação transmitida como característica cultural. Referências para exemplos aplicados ao PCV podem ser encontrados em (PRESTES, 2006).

Algoritmos Transgenéticos têm como base o processo de Endossimbiose / Quorum Sensing, dos quais, a evolução não ocorre pela troca de informações dentro da população de cromossomos, mas sim, entre a população de cromossomos e uma população de vetores transgenéticos (transportam uma ou mais cadeias de informações e disponibilizam métodos que definem sua atuação na população de cromossomos). Por definição, Schmidt descreve a endossimbionte como qualquer organismo que vive no interior do corpo ou das células de outro organismo, realizando uma relação ecológica designada como endossimbiose. Enquanto que, Quorum Sensing, reflete a habilidade das bactérias de comunicação e coordenação através de sinais moleculares. Referências para exemplos aplicados ao PCV podem ser encontrados em (PRESTES, 2006).

A Otimização por Nuvem de Partículas ou, do inglês Particle Swarm Optimization (PSO), é uma metodologia baseada no comportamento do voo em bando dos pássaros (nuvem ou população), em que uma função mediria os esforços de cada pássaro (partícula) em manter uma distância ótima de seu vizinho. É composto por uma função objetivo que avalia e direciona partículas no espaço de soluções, conforme velocidade de cada uma. Referências para exemplos aplicados ao PCV podem ser encontrados em (PRESTES, 2006).

Atualmente, um dos maiores desafios é o da TSP World Tour (TSP, 2010), com 1.904.711 cidades. Em 4 de maio de 2010, Helsgaun, utilizando uma variante do algoritmo LK-H encontrou uma turnê com o comprimento de 7,515,796,609. Este relato, supera o algoritmo LK-NYYY. O menor limite inferior para o problema é de 7,512,218,268 encontrado pelo algoritmo Concorde TSP (5 de junho de 2007).

3. Algoritmos Genéticos

Algoritmos Genéticos (AG), foram introduzidos pelo matemático John H. Holland (1975), e demonstram grande eficiência na resolução de problemas complexos. São inspirados na seleção natural de Darwin (processo de evolução dos seres vivos). "A técnica, consiste em explorar o espaço de busca através de uma população de soluções viáveis (cromossomos), que evolui a cada geração por meio de operadores genéticos de cruzamento e mutação." (GUEDES, 2005)

Os cromossomos podem ser representados de formas diferentes. Como exemplo,

no PCV, um conjunto de 1 a 8 cidades, {3, 8, 7, 2, 1, 5, 4, 6} representa uma das soluções viáveis.

O PCV, por se tratar de um problema complexo, que cresce exponencialmente conforme a quantidade de parâmetros, não permite o cálculo de todos os caminhos em tempo aceitável. É justamente neste ponto, que os Algoritmos Genéticos se mostram como ferramentas poderosas no intuito de obter a solução em tempo plausível.

De modo geral, além de proporcionarem simplicidade e redução no espaço de busca, AG são aplicados em situações como: modelo matemático desconhecido/impreciso e em funções lineares e ou não-lineares (COSTA et al., 2003).

1. **Inicialização:** gerar uma população inicial de n cromossomos, aleatoriamente, e determinar o fitness de cada cromossomo;
2. **Nova população:** criar uma nova população através da aplicação das seguintes etapas:
   a) **Seleção:** selecionar dois cromossomos-pais da população atual de acordo com sua fitness;
   b) **Crossover:** cruzamento dos pais para formar novos indivíduos (filhos);
   c) **Mutação:** aplicar mutação nos novos indivíduos;
3. **Avaliar nova população:** Calcular a fitness de cada cromossomo da população recém-gerada;
4. **Teste de parada:** se condição de parada satisfeita: finalizar retornando a melhor solução encontrada. Caso contrário, voltar ao passo 2.

Figura 1. Pseudo código para AG (GUEDES, 2005)

Para a metodologia de seleção foram utilizadas duas formas: roleta e *ranking*. Sendo esta última, a técnica aplicada nos experimentos aqui apresentados. Quanto ao cruzamento, conforme indicado na literatura e, por proporcionar melhores resultados, foi empregado o de dois pontos.

3.1. Implementação

A ferramenta utilizada na implementação será Java, por se tratar de uma linguagem amplamente difundida (materiais como livros, bibliotecas, api's em diversas áreas de atuação), com qualquer tipo de aplicação (web, desktop, servidores, mainframes, jogos, aplicações móveis, chips de identificação, etc.). Embora, se tratar de uma linguagem híbrida (compilada + interpretada), atualmente, possui desempenho equivalente à outras linguagens com alto desempenho, que é obtido graças ao compilador Just-In-Time (JIT) e técnicas que detectam pontos de maior frequência. Em uma parte (de maior frequência) é gerado código nativo do sistema operacional, enquanto demais partes, são interpretadas. (SDN, 2010) (PAMPLONA, 2010)

Para atender às necessidades de um sistema baseado em AG's, tem-se como principais requisitos do sistema, a passagem de parâmetros: tamanho da população, número de gerações, número de vezes que será rodado o algoritmo, o arquivo da TSPLIB a ser avaliado, probabilidade de cruzamento e de mutação. Com o objetivo de atender estas características que definem/interferem diretamente no comportamento, é

que o sistema foi projetado para aceitar a linha de comando segundo a sintaxe:

java -jar tsp.jar [population_size] [max_generations] [max_runs] [file.tsp] [n_par] [log] [prob_cross] [prob_mutate]

Os argumentos devem respeitar e ou levar em consideração os seguintes critérios constantes na literatura:

- population_size: tamanho da população, afeta o desempenho/eficiência. População pequena o desempenho cai devido ao espaço de busca reduzido.
- max_generations: número máximo de gerações por população
- max_runs: número de vezes que o algoritmo irá rodar
- file.tsp: arquivo da TSPLIB
- n_par: valores positivos maior ou igual a 0. Para rodar sem paralelismo utilizar 0 ou 1. Este valor representa o número de *threads* simultâneas.
- log: armazena arquivo de *log* contendo estatísticas da resolução. Utilizar "none" para não gerar.
- prob_cross: Probabilidade de cruzamento. Quanto maior for esta taxa, mais rapidamente novas estruturas serão introduzidas na população. Mas se esta for muito alta, estruturas com boas aptidões poderão ser retiradas mais rapidamente. Valores de referência são entre 60% e 65%. (MIRANDA, 2007)
- prob_mutate: Probabilidade de mutação. Uma baixa taxa de mutação previne que uma dada posição fique estagnada em um valor, além de possibilitar que se chegue em qualquer ponto do espaço de busca. Com uma taxa muito alta a busca se torna essencialmente aleatória. Os valores de referência são entre 0,1% e 5%. (MIRANDA, 2007)

O tamanho da população e o número de gerações dependem da complexidade de otimização do problema e devem ser determinados experimentalmente. Ambos definem o espaço de busca a ser coberto. Visto a importância da escolha destes parâmetros, foram criados AG's capazes de avaliar os parâmetros de outros AG's.

3.2. Argumentos de otimização da JVM

Devido à tecnologia utilizada (Java - linguagem hibrida), existem parâmetros que passados para a JVM interferem diretamente no comportamento do algoritmo. O principal foco discutido aqui será o de comandos que definem o desempenho/recursos que serão alocados.

Tabela 1. Alguns argumentos de otimização da JVM

Argumento	Descrição
-server	Melhora o desempenho para uso intensivo da CPU. Utilizado para aplicações que requerem alto desempenho.
-Xoptimize	otimização do compilador JIT
-Xms	define heap inicial da JVM.
-Xmx	define o tamanho máximo do heap da JVM.

Sintaxe resumida:

java [options_jvm] -jar tsp.jar [arguments_tsp]

Exemplos:

java -server -Xms256m -Xmx512m -Xoptimize -jar tsp.jar 70 100 10 berlin52.tsp 2 none 0.6 0.04

java -server -Xms256m -Xmx512m -Xoptimize -jar tsp.jar 70 100 10 berlin52.tsp 2 berlin52.txt 0.6 0.04

Os argumentos descritos aqui, são apenas alguns dos citados no artigo de (ORT, 2001), sendo estes, definidos como um dos melhores resultados segundo testes realizados e apresentados pelo mesmo. Vale ressaltar que tais parâmetros podem e devem ser alterados conforme necessidade/complexidade/tamanho do problema. Principalmente os comandos relativos ao heap (memória alocada) para execução do algoritmo.

Em situações em que a memória alocada não for suficiente, será apresentado o seguinte erro: "Exception in thread "main" java.lang.OutOfMemoryError: Java heap space". Este problema pode ser resolvido com o argumento -Xmx, respeitando o limite de memória disponível pelo computador.

Maiores informações sobre argumentos da JVM podem ser encontrados em (ORT, 2001; ORACLE, 2010).

3.3. Estrutura do algoritmo

Procurando respeitar a orientação a objetos, foram estabelecidos 10 pacotes de código fonte para armazenar as classes de acordo com o conteúdo de modo a proporcionar futuras implementações.

Tabela 2. Pacotes de código fonte e separação segundo seus conteúdos

Pacote	Conteúdo
avaliation	Avaliação (fitness)
consts	Constantes utilizadas no sistema
exception	Tratamento de erros
file	Interpretadores dos arquivos fornecidos pela TSPLIB e arquivo de log.
graph	Grafos
interfaces	Definição das principais interfaces do sistema
operations	Operações envolvidas em AG
solver	Implementa uma metodologia na solução do TSP
tsp	Classes diretamente relacionadas ao TSP
utilitys	Classes contendo tratamento de métodos em geral que são úteis ao sistema

Tabela 3. Descrição das classes separadas por pacote

Pacote	Classe	Descrição
avaliation	Fitness	Avaliar aptidão de cromossomo
consts	TSPConsts	Constantes utilizadas no sistema
exception	TSPException	Tratamento de erros pelo sistema
file	LogFile	Arquivo de log com estatísticas da resolução.
	TSPFileParser	Interpretador de arquivo da TSPLIB
	TSPOptFileParser	Interpretador de arquivo da TSPLIB que contém o melhor tour
	TSP_EUC_2D	Converte um vetor de coordenadas euclidianas 2D em um TSPGraph
	TSP_EUC_2D_Float	Converte um vetor de coordenadas euclidianas 2D em um TSPGraphFloat
	TSP_EUC_2D_Integer	Converte um vetor de coordenadas euclidianas 2D em TSPGraphInteger
	TSP_GEO	Converte um vetor de coordenadas geográficas em um TSPGraph
	TSP_GEO_Float	Converte um vetor de coordenadas geográficas em um TSPGraphFloat
	TSP_GEO_Integer	Converte um vetor de coordenadas geográficas em um TSPGraphInteger
graph	TSPGraph	Matriz de distâncias entre cada cidade de tipo genérico
	TSPGraphFloat	Matriz de distâncias entre cada cidade de tipo Float
	TSPGraphInteger	Matriz de distâncias entre cada cidade de tipo Integer
interfaces	Chromosome	Interface para o tipo cromossomo
	Individual	Interface para o tipo indivíduo
	OperationAG	Interface para operações envolvidas em AG
	Problem	Interface para o problema TSP
	Solver	Interface para solucionar o problema TSP
operations	Copy	Operação de cópia
	Crossover	Operação de cruzamento
	Mutate	Operação de mutação
	Selection	Operação de seleção
solver	TSPSolver	Implementa a interface Solver com a lógica de AG
	TSPSolverPar	Paralelização do TSPSolver
tsp	Main	Responsável por executar o algoritmo e passar os argumentos
	TSP	Implementa o problema TSP
	TSPChromosome	Define o tipo cromossomo
	TSPCoordinate	Define o tipo coordenada
	TSPIndividual	Define o tipo indivíduo
	TSPPopulation	Define o tipo população
	TSPReader	Interpretador dos argumentos passados em linha de comando
utilitys	Points	Implementa geração de pontos para cruzamento e mutação aleatoriamente
	Randomic	Define a forma de geração de valores aleatórios
	Text	Trata de operações com texto

3.4. Paralelismo

No Java, diferentemente de outras linguagens, possui primitivas de multithreading como parte da própria linguagem e de suas bibliotecas. O que facilita a manipulação de segmentos de maneira portável entre plataformas. (DEITEL, 2006)

Em computadores com mais de um núcleo, segmentos podem ser distribuídos entre cada um, fazendo com que o aplicativo opere de forma mais eficiente.

No Java, podem ser citados as classes: Thread, Runnable, ThreadGroup, Callable, Future, Executor; sendo as três primeiras, parte da biblioteca java.lang e as três últimas da java.util.concurrent.

Descrição das classes:

- Runnable: interface contendo a assinatura do método run(), responsável pela execução de uma ou mais tarefas, e que deve ser sobrescrito. Esta deve ser passada como argumento na criação de uma Thread para que seu método principal run() seja executado.
- Thread: classe que permite criar linhas de execuções simultâneas.
- ThreadGroup: classe responsável por gerenciar um grupo de threads.
- Callable: interface similar à Runnable, exceto pelo fato de permitir o retorno de um valor. A assinatura do seu método principal é <Object>call(), onde <Object> define um tipo genérico e, é utilizado para definir o tipo a ser retornado pela função.
- Future: classe que trata o resultado de uma computação assíncrona, implementa métodos para verificar o término dos cálculos. Seu principal método é o get() que retorna um tipo definido pelo programador. Em seu método construtor deve-se passar como argumento o objeto de tipo Callable.
- Executor: Framework para gerenciar threads. Considerado mais eficiente que ThreadGroup, pois podem reutilizar threads existentes para eliminar a sobrecarga de criação de novas threads. Permitem a utilização de Runnable, Thread, Callable e Future; além de permitir que seja determinado o número de tarefas que serão executados simultaneamente.

Foram realizados testes em cima do paralelismo, dentre os quais obteve-se resultados satisfatórios com a biblioteca java.util.concurrent. O processo de paralelização consistiu basicamente em criar uma nova versão da classe Solver, em que foi instanciado um único objeto do tipo Executor para submeter as linhas de execução, uma classe que implementa a interface Callable na construção de uma nova população, instâncias da classe Future para coletar os resultados e a sincronização de métodos *set()* e *get()* do melhor indivíduo na classe que trata o problema.

Ainda foi necessário a adequação de parâmetros na linha de comando para permitir que o algoritmo pudesse ser executado ou não de forma paralela e a determinação do número de processos simultâneos.

Os testes apontam que o número ideal de processos seja equivalente ao de núcleos encontrados no computador. Quantidades superiores fazem com que o algoritmo leve um tempo maior na sua execução.

4. Experimentos

Variados experimentos com problemas de tamanhos diferentes, bem como a aplicação de parametrização diversificada foram aplicadas. Foram utilizados parâmetros de cruzamento e mutação conforme recomendado na literatura e previamente citados na seção 3.1. O computador utilizado para os testes foi um Core 2 Duo – 1.66 Ghz com 2 GB de memória, sendo o sistema operacional utilizado o LINUX.

Tabela 4. Resultados do AG sem paralelização

PROBLEMA	P	G	R	XC	XM	T(ms)	S	M
berlin52.tsp	1000	1000	10	0,60	0,04	58060	7585	7542
gr666.tsp	400	50000	10	0,60	0,04	13842457	705258	293611
ulysses16.tsp	70	80	10	0,60	0,04	1545	6756	6756
ulysses22.tsp	180	180	10	0,60	0,04	4296	6912	6912

Tabela 5. Resultados do AG com paralelização

PROBLEMA	P	G	R	XC	XM	T(ms)	S	M
berlin52.tsp	1000	1000	10	0,60	0,04	32991	7843	7542
gr666.tsp	400	50000	10	0,60	0,04	7084493	750180	293611
ulysses16.tsp	70	80	10	0,60	0,04	1011	6756	6756
ulysses22.tsp	180	180	10	0,60	0,04	2969	6912	6912

P	Tamanho da população
G	Número de gerações
R	Número de execuções do algoritmo
XC	Probabilidade de cruzamento
XM	Probabilidade de mutação
T	Tempo de execução (ms)
S	Solução obtida neste trabalho
M	Melhor solução encontrada na literatura

Figura 2. Legenda para Tabelas 4 e 5

Os resultados apresentados nas tabelas 4 e 5 demonstram que o algoritmo se apresenta de forma eficiente, conseguindo atingir resultados ótimos (ulysses16, ulysses22) e aceitáveis (berlin52) – cerca de 0,52% maior que o melhor percurso.

A performance do algoritmo pode ser avaliada segundo a análise de programas paralelos onde os parâmetros a serem considerados são: (MESQUITA, 2010)

- Tempo de execução: tempo decorrido desde o início do processamento até o término de execução do último processador. Durante este processo, um