UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Bruno Sérgio Cardoso Vieira

Escalonamento multiobjetivo de processos

Uberlândia, Brasil
2017

1 Introdução

Aluno(a): Bruno Sérgio Cardoso Vieira
Matrícula: 11311BCC009
Tema abordado: Escalonamento multiobjetivo de processos
Professor Orientador: Paulo Henrique Ribeiro Gabriel

1.1 Visão Geral

O escalonamento de processos se tornou uma tarefa fundamental para um melhor uso de sistemas computacionais distribuídos [3]. Diversas soluções têm sido propostas para tratar esse problema, dentre as quais se destacam algoritmos heurísticos [1, 12]. Nesse contexto, os algoritmos genéticos (AGs) são um ramo dos algoritmos heurísticos que adotam conceitos biológicos para otimizar determinadas funções [11, 13].

No caso do problema de escalonamento, duas funções têm sido comumente empregadas: *i*) *Makespan*, que avalia o tempo máximo de espera para a execução de uma aplicação; e *ii*) o *Flowtime* que mede o tempo médio gasto na execução de processo, a partir do momento que esse é lançado no sistema até sua total execução. Na literatura, essas duas funções têm sido avaliadas em conjunto, combinadas por meio de somas ponderadas [12]. Poucos trabalhos, no entanto, as avaliam individualmente, por meio de algoritmo de otimização multiobjetivos [13]. Tal limitação motiva o desenvolvimento deste trabalho.

1.2 Objetivo

O objetivo geral deste trabalho é implementar o algoritmo genético multiobjetivo para o problema de escalonamento de processos, considerando as métricas *Makespan* e *Flowtime*, e verificar se uma implementação multiobjetivo de um AG, que considera as duas funções separadamente, pode trazer bons resultados quando comparada com implementações que combinam os objetivos em somas ponderadas.

1.3 Justificativa

A literatura nessa área, em geral, não avalia a influência da otimização entre as métricas. A utilização de funções de ponderação podem não ser adequadas, pois tal correlação não é considerada. Foi observado que *Makespan* e *Flowtime* são objetivos contraditórios, ou seja, a otimização de um pode não resultar na otimização do outro [2]. Com esse trabalho, pretende-se avaliar tal correlação e analisar o comportamento de um algoritmo multiobjetivo.

1.4 Método de Pesquisa

A propósito de cumprir o objetivo proposto, será implementado um algoritmo genético multiobjetivo, considerando a minimização da métrica *Makespan* e *Flowtime*. Posteriormente, o algoritmo será executado sobre instâncias do problema de escalonamento de processos. Em seguida, os resultados do algoritmo implementado será comparado aos resultados de algoritmos genéticos não-multiobjetivo. Dessas comparações, será possível extrair conclusões a fim de verificar a relação entre as métricas e o desempenho do algoritmo multiobjetivo.

1.5 Resultados Esperados

Espera-se obter resultados suficientes para avaliar o comportamento de um algoritmo genético multiobjetivo sobre o problema de escalonamento de processos de forma a apontar a melhoria das soluções encontradas quando o algoritmo considera, simultaneamente, as métricas *Makespan* e *Flowtime*, em comparação aos algoritmos utilizados atualmente na literatura.

2 Referencial Teórico

2.1 Algoritmo Genético

Algoritmos genéticos (AGs) são métodos de otimização e busca inspirados nos mecanismos de evolução dos seres vivos [11], utilizando conceitos como mutação, seleção natural, hereditariedade e reprodução. O algoritmo genético avalia um conjunto de indivíduos por meio de uma função objetivo (também chamada de função de aptidão), e utiliza um processo de combinação entre os indivíduos melhor avaliados para gerar novos indivíduos com aptidão possivelmente melhor a cada iteração. Dessa forma, obtém-se resultados cada vez mais próximos da melhor solução para o problema (ponto de máximo ou mínimo da função objetivo).

2.1.1 Cromossomo

O cromossomo é a estrutura de dados utilizada para representar um indivíduo. Um cromossomo armazena as características do indivíduo que serão avaliadas pelo AG. O conjunto de todas as possíveis configurações de cromossomo constitui o espaço de busca [11]. Geralmente, os cromossomos são representados por vetor ou cadeia de bits. A Figura 1 exemplifica uma possível estrutura para representar um cromossomo utilizando uma cadeia de bits. Supõe-se indivíduos com 5 características, cada uma delas sendo avaliadas com valores de 0 a 15. Os valores dessas características são concatenados como valores binários de 4 bits (quantidade suficiente de bits para representar a faixa de possíveis valores de uma característica). O cromossomo resultante é uma cadeia de 20 bits. A informação original pode ser facilmente obtida realizando o processo inverso, agrupando 4 bits e recuperando seu valor decimal.

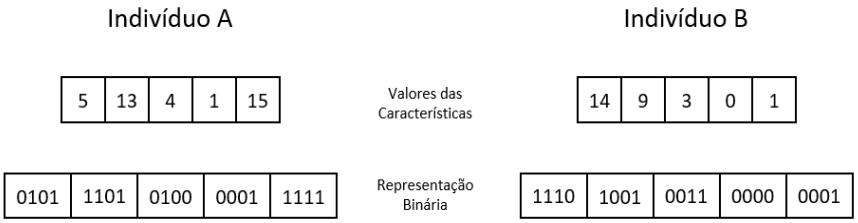

Figura 1: Cromossomo representado por cadeia de bits

2.1.2 Aptidão

A aptidão (ou *fitness*) é o valor associado à um cromossomo que representa sua qualidade, calculado por uma função objetivo. É por meio dessa avaliação que o algoritmo genético seleciona os melhores indivíduos de uma população e os prioriza no processo de combinação. Na Figura 2, a avaliação é feita simplesmente por converter a cadeia de bits para seu valor decimal correspondente.

2.1.3 Algoritmo genético típico

Um AG típico se inicia pela construção de uma população inicial, composta por cromossomos aleatoriamente gerados. As aptidões de cada cromossomo são calculadas e atribuídas. Um operador de seleção descarta os cromossomos de menor aptidão, enquanto que os cromossomos de maior aptidão podem ter suas estruturas modificadas por operadores de permutação e mutação, gerando

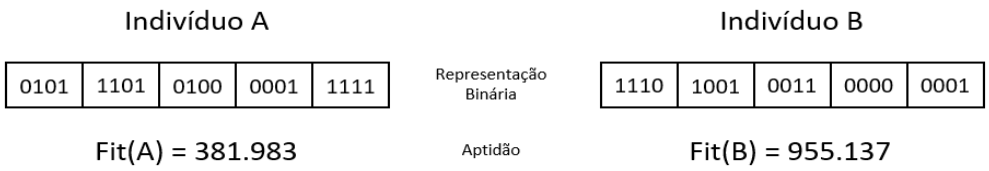

Figura 2: Avaliação da aptidão de indivíduos

novos indivíduos, que são adicionados à população para a próxima geração. O processo é repetido sobre a nova população até que um critério seja atingido, como número de iterações ou valor da melhor aptidão encontrada [11]. A estrutura de um algoritmo genético típico é mostrada na Figura 3.

```
/* Seja P(t) a população de cromossomos na geração t." */
t ← 0
inicializar P(t)
avaliar P(t)
ENQUANTO o critério de parada não for satisfeito FAÇA
    t ← t + 1
    selecionar P(t) a partir de P(t − 1)
    aplicar permuta sobre P(t)
    aplicar mutação sobre P(t)
    avaliar P(t)
FIM ENQUANTO
```

Figura 3: Estrutura de um algoritmo genético típico [11]

1) Operador de seleção: a partir de uma lista contendo os cromossomos da população e seus respectivos valores de aptidão, o AG é capaz de selecionar os indivíduos de maior aptidão. A escolha do critério de seleção é importante para o desenvolvimento do algoritmo genético, pois influencia a velocidade de convergência e qualidade da solução encontrada. Dentre as diversas alternativas propostas para essa etapa do algoritmo estão o algoritmo da roleta (ou aleatório), seleção por torneio (ou competição) e seleção por classificação [2].

    a) Roleta: neste algoritmo, os indivíduos são selecionados aleatoriamente de que os indivíduos de maior aptidão têm maior probabilidade de serem selecionados.

    b) Torneio: são criados grupos de indivíduos, dentre os quais apenas os indivíduos de maior aptidão são selecionados.

    c) Classificação: semelhante ao algoritmo da roleta, tem a característica de melhor distribuir a probabilidade de seleção para indivíduos de menor aptidão.

2) Operador de permutação (ou cruzamento): é a etapa do algoritmo genético que produz novos indivíduos a partir dos indivíduos selecionados na etapa de seleção. Nessa etapa, trocas de porções da estrutura dos cromossomos são realizadas de acordo com algum critério, dentre eles o cruzamento de um ponto, de dois pontos e uniforme [2]. A Figura 4 exemplifica a forma de operação dos três critérios citados, utilizando a estrutura de cadeia de bits.

3) Operador de mutação: executa alterações aleatórias nos cromossomos da população, com baixa probabilidade. O objetivo desse operador é manter a diversidade da população e evitar a convergência prematura da população. A aleatoriedade dessa alteração causa a geração de indivíduos em outras regiões do espaço de busca, fazendo com que uma maior área seja explorada e, consequente, mais indivíduos sejam avaliados no processo de convergência. A Figura 5 apresenta uma operação de mutação sobre uma cadeia de bits.

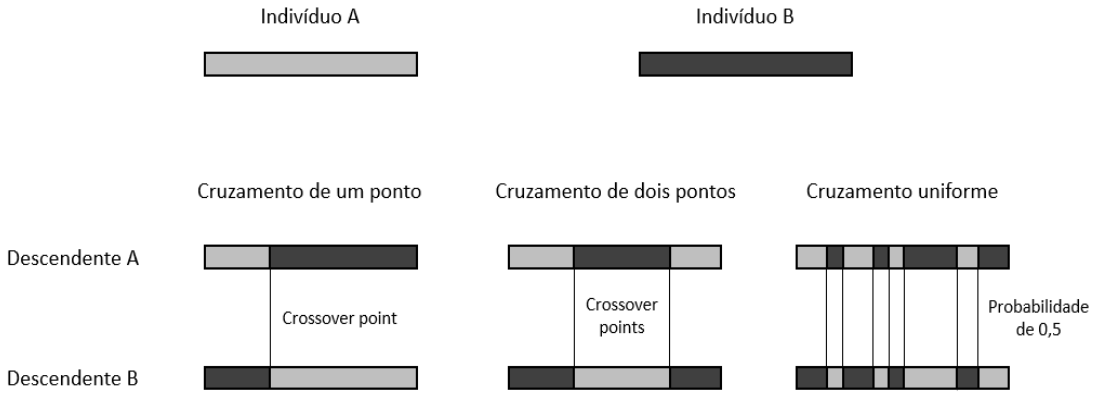

Figura 4: Exemplos de permutação em cadeia de bits

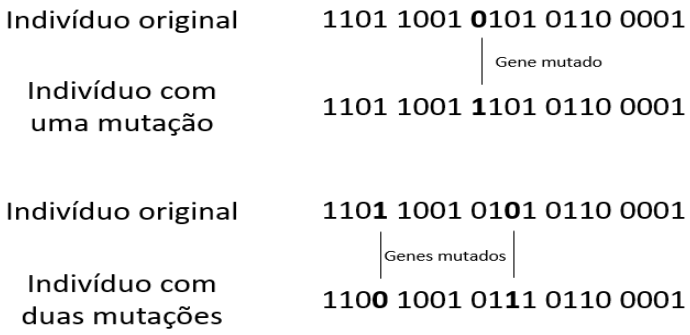

Figura 5: Exemplo de mutação sobre cadeia de bits

2.1.4 Convergência prematura

Ocorre quando indivíduos de alta aptidão (porém não ótima) dominam a população antes que indivíduos realmente ótimos sejam encontrados pelo algoritmo, ou seja, o espaço de busca do AG foi explorado localmente. Dessa forma, o algoritmo converge para uma solução local. Algumas técnicas podem ser utilizadas para prevenir ou reduzir esse efeito, como o aumento da taxa de mutação e limitação de descendentes de um cromossomo [11].

2.1.5 Algoritmo Genético Multiobjetivo

Algoritmos genéticos multiobjetivo são uma derivação dos algoritmos genéticos capazes de resolver um problema constituído por mais de uma função objetivo, ou seja, há a necessidade de avaliar as soluções em mais de um aspecto.

Algumas alterações são feitas no algoritmo para que seja possível manipular os vários objetivos. Por exemplo, o algoritmo pode manter duas populações distintas para armazenar, separadamente, os indivíduos selecionados de acordo com cada objetivo. Nota-se, também, as adições ao algoritmo que o permitam interagir e relacionar essas populações.

Outro ponto importante em um algoritmo genético multiobjetivo é a relação entre os objetivos, que pode ser hierárquica ou simultânea [9]. Na abordagem simultânea, todos os objetivos são otimizados simultaneamente. Na abordagem hierárquica, os objetivos são ordenados de acordo com sua importância. O processo otimiza o objetivo mais importante até que não seja mais possível. Então, o algoritmo passa a otimizar o próximo objetivo de forma a não deteriorar os valores obtidos na otimização do objetivo anterior.

2.2 Escalonamento de Processos

Escalonamento de processos é o nome dado à atividade de atribuir um conjunto de processos a um conjunto de elementos de processamento [3]. A solução é avaliada sobre um critério de otimização, que indica a qualidade da solução em relação a algum aspecto do problema.
Alguns termos comumente utilizados no problema de escalonamento de processo [12].

a) Tarefa: Unidade indivisível de trabalho

b) Processo: Conjunto de tarefas, dependentes ou independentes. Tarefas dependentes devem ter sua ordem de execução consideradas pelo escalonador, enquanto que tarefas independentes não possuem pré-requisito de execução.

c) Aplicação: Software responsável por resolver um problema computacional. Pode ou não ser dividido em vários processos.

d) Recurso: Unidade de processamento de tarefas e processos. Cada recurso possui suas próprias características, como velocidade de processamento e memória.

2.2.1 Métricas de Desempenho

A formalização do problema de escalonamento de processos utilizada neste trabalho segue a definição feita por Gabriel [5].Inicialmente, considera-se um conjunto A de aplicações paralelas e uma função $tam(a_k)$ que define o número de processos contidos em uma aplicação $a_k \in A$. Cada aplicação contém um número diferente de processos. Seja um conjunto P, que contém todos os processos de todas as aplicações. Temos que o número de elementos (ou cardinalidade) de P é dada por

$$|P| = \sum_{k=1}^{|A|} tam(a_k)$$

Processos em P são alocados sobre um conjunto de computadores, denotado por V, em que cada computador v_j E V pode ter diferentes capacidades: velocidade de processamento, tamanho e tempo de acesso à memória principal, capacidade e tempo de acesso ao disco rígido, etc. Os computadores em V estão conectados por diferentes canais de comunicação. Assim, esse ambiente pode ser modelado por um grafo não-dirigido $G = (V, E)$, em que cada vértice $v_j \in |V|$ representa um computador e os canais de comunicação entre pares de vértices são descritos por arestas $(v_j, v_i) \in E$ do grafo G.

Diversas métricas são utilizadas para avaliar a solução produzida pelo escalonador. Dentre as mais usadas, estão *Makespan*, *Flowtime*, *Load Balancing* (balanceamento de carga), *Utilization* (utilização de recursos) entre outras [12]. Apenas os critérios *Makespan* e *Flowtime* serão avaliados neste trabalho. Considere que C_j, para $j = 1, 2, ... |V|$, denota o tempo que cada computador permanece ocupado para executar os processos atribuídos ao recurso j.

Makespan é o intervalo de tempo total entre o início e o término da execução de um processo $p_i \in P$, ou seja, o maior valor de C_j [4]. Dessa forma, pode-se definir o *Makespan* por

$$M = \max_{(1 \leq j \leq |V|)} C_j$$

Já a métrica *Flowtime* é definida pela soma de todos os tempos gastos para executar todos os processos pi E P [2]), sendo definida por

$$F = \sum_{j=1}^{|V|} C_j$$

O objetivo do escalonador é minimizar ambas as métricas. A minimização do *Makespan* indica que os recursos permanecem menos tempo ocupados, enquanto que a minimização do *Flowtime* indica que todos os processos foram executados em um tempo menor. Porém, a minimização de uma não implica, necessariamente, na minimização da outra [2].

2.2.2 *ETC Model*

O modelo ETC (*Expected Time to Compute*) é um modelo, em formato de matriz, que armazena uma estimativa do tempo de finalização esperado de cada processo em cada recurso [8]. A matrix ETC é uma matrix $n_x m$, onde n é o número de processos e m é o número de recursos. Dessa forma, uma linha da matriz contém o tempo de execução de um processo i em cada um dos recursos disponíveis, enquanto que uma coluna da matriz contém o tempo de execução de todos os processos em um determinado recurso j. *ETC[i][j]* contém o tempo necessário para a tarefa i ser completamente processada pelo recurso j. A Figura 6 exibe uma matriz ETC para uma grade computacional com 5 recursos, para qual é atribuída o processamento de uma aplicação com 5 tarefas.

	Recurso 1	Recurso 2	Recurso 3	Recurso 4	Recurso 5
Tarefa 1	10	13	7	5	10
Tarefa 2	5	5	4	10	15
Tarefa 3	12	8	13	5	12
Tarefa 4	15	20	18	16	20
Tarefa 5	6	10	8	12	14

Figura 6: Exemplo de matriz ETC.

A partir dessa matriz, é possível definir os valores de *Makespan* e *Flowtime*. Para avaliar o *makespan*, observe que a Tarefa 4 possui o maior tempo de execução. Dessa forma, não é possível obter resultados menores que 15. Logo, qualquer combinação do escalonamento das tarefas que obtenha tempo máximo de 15 é uma solução ótima para a minimização do *makespan*. A Figura 7 exibe uma dessas possíveis soluções.
Já a Figura 8 exibe uma possível solução para a minização da métrica *flowtime*. O tempo de finalização da tarefa 3 é 13, visto que a tarefa 2 é executada primeiramente. Assim, o valor da métrica *flowtime* para o exemplo é 47.

Tarefa 1	Recurso 4
Tarefa 2	Recurso 3
Tarefa 3	Recurso 4
Tarefa 4	Recurso 1
Tarefa 5	Recurso 3

Makespan = 15

Figura 7: Possível solução ótima para a métrica *makespan* para a matrix ETC da figura 6.

Tarefa 1	Recurso 3
Tarefa 2	Recurso 2
Tarefa 3	Recurso 2
Tarefa 4	Recurso 4
Tarefa 5	Recurso 1

Flowtime = 7 + 5 + 13 + 16 + 6 = 47

Figura 8: Possível solução ótima para a métrica *flowtime* para a matrix ETC da figura 6.

3 Estado da Arte

3.1 Trabalhos Correlatos

Esta seção expõe alguns trabalhos desenvolvidos utilizando algoritmos genéticos no problema escalonamento de processos. A simplicidade de implementação dos algoritmos genéticos faz com que esses algoritmos sejam amplamente estudados e utilizados em trabalhos nesse contexto [1].

Braun et al. [1] mostrou que o algoritmo genético proposto em seu trabalho obteve melhores resultados em comparação a onze heurísticas para o problema de escalonamento de aplicações independentes. O algoritmo obteve melhor eficiência e qualidade nas soluções encontradas

Izakian and Abraham [6] analisa o desempenho de 6 algoritmos meta-heurísticos diferentes para a minimização das métricas *Makespan* e *Flowtime* para processos independentes. O trabalho concluiu que a heurística Min-min, que utiliza uma métrica que avalia a tarefa que pode ser completada mais rapidamente, como o algoritmo de melhor resultado. Embora AGs não tenham sido considerados, esse trabalho contém resultados de algoritmos comumente utilizados na literatura, que serão utilizados para comparação do algoritmo multiobjetivo proposto neste trabalho.

Kolodziej et al. [9] formaliza o problema de escalonamento de processos utilizando um algoritmo genético multiobjetivo hierárquico sobre as métricas *Makespan* e *Flowtime*, porém com o intuito de minimizar, também, o consumo de energia. Os experimentos indicam que os dois algoritmos genéticos propostos obtiveram bons resultados.

Kolodziej and Khan [8] utiliza algoritmo genético multiobjetivo no problema de escalonamento de processos, porém avalia apenas *Makespan* como objetivo dominante e *Flowtime* como objetivo secundário, ou seja, utilizando a abordagem hierárquica. A análise não considera a otimização abordada simultaneamente. Conclui-se que o algoritmo proposto pelo trabalho obteve maior eficiência em comparação a algoritmos mono-objetivo e estratégias híbridas.

Em outra abordagem, Izakian et al. [7] compara a otimização de grupos densos de partículas com algumas abordagens utilizadas na literatura, incluindo algoritmos genéticos, para otimização das métricas *Makespan* e *Flowtime*. Os resultados do algoritmo proposto supera os resultados dos algoritmos comparados, na maioria dos casos.

Também em outra abordagem, Kromer et al. [10] propõe um algoritmo baseado em evolução diferencial. Tal algoritmo obteve resultados considerados competitivos, e uma otimização da população inicial melhorou seus resultados.

4 Cronograma

As futuras etapas do desenvolvimento dessa pesquisa devem seguir planejamento de atividades indicado na tabela 1.

Tarefa	*Periodo*
Estudo de implementações de algoritmos genéticos	14/08 a 01/09
Implementação do algoritmo genético multiobjetivo	04/09 a 06/10
Execução do algoritmo nas instâncias do problema	09/10 a 20/10
Comparação dos resultados com outros algoritmos	23/10 a 03/11
Análise dos resultados	06/11 a 10/11
Escrita da monografia	13/11 a 08/12

Tabela 1: Cronograma de atividades

Assinatura do Orientador:

Referências

[1] Tracy D. Braun, Howard Jay Siegel, Noah Beck, Lasislau L. Bölöni, Muthucumara Maheswaran, Albert I. Reuther, James P. Robertson, Mitchell D. Theys, Bin Yao, Debra Hensgen, and Richard F. Freund. A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems. *Journal of Parallel and Distributed Computing*, 61(6):810–837, June 2001.

[2] Javier Carretero, Fatos Xhafa, and Ajith Abraham. Genetic algorithm based schedulers for grid computing systems. *International Journal of Innovative Computing, Information and Control*, 3(6):19, 2007.

[3] Steve J. Chapin and Jon B. Weissman. Distributed and multiprocessor scheduling. In Allen B. Tucker, editor, *Computer Science Handbook*, chapter 88, pages 88–1–88–19. CRC Press, 2004.

[4] Fangpeng Dong and Selim G. Akl. Scheduling algorithms for grid computing: State of the art and open problems. Technical report, School of Computing, Queen's University, 2006.

[5] Paulo Henrique Ribeiro Gabriel. *Uma abordagem orientada a sistemas para otimização de escalonamento de processos em grades computacionais.* PhD thesis, Universidade de São Paulo, 2013.

[6] Hesam Izakian and Ajith Abraham. Comparison of heuristics for scheduling independent tasks on heterogeneous distributed environments. *International Joint Conference on Computational Sciences and Optimization*, page 5, 2009.

[7] Hesam Izakian, Behrouz Tork Ladani, Ajith Abraham, and Vaclav Snasel. A discrete particle swarm optimization approach for grid job scheduling. *International Journal of Innovative COmputing, Information and Control*, 6(9):15, 2009.

[8] Joanna Kolodziej and Samee Ullah Khan. Multi-level hierarchic genetic-based scheduling of independent jobs in dynamic heterogeneous grid environment. *Information Sciences*, page 19, 2012.

[9] Joanna Kolodziej, Samee Ullah Khan, and Fatos Xhafa. Genetic algorithms for energy-aware scheduling in computational grids. *Internationa COnference on P2P, Parallel, Grid, Cloud and Internet Computing*, page 8, 2011.

[10] Pavel Kromer, Vaclav Snasel, Jan Platos, Hesam Izakian, and Ajith Abraham. Scheduling independent tasks on heterogeneous distributed environments by differential evolution. *International Conference on Intelligent Networking and Collaborative Systems*, page 5, 2009.

[11] E. G. M. Lacerda, A. C. P. L. de Carvalho, and T. B. Ludermir. Um tutorial sobre algoritmos genéticos. *Revista de Informática Teórica e Aplicada*, 9:7–39, 2002.

[12] Fatos Xhafa and Ajith Abraham. Computational models and heuristic methods for grid scheduling problems. *Future Generation Computer Systems*, 26(4):608–621, 2010.

[13] E. Zitzler, M. Laumanns, and S. Bleuler. A tutorial on evolutionary multiobjective optimization. Technical report, Swiss Federal Institute of Technology Zurich, Glorastrasse 35, CH-8092 Zurich, Switzerland, 2003.