11º FEPEG FÓRUM ENSINO • PESQUISA EXTENSÃO • GESTÃO
UNIVERSIDADE, SOCIEDADE E POLÍTICAS PÚBLICAS
ISSN: 1806-549X
Realização: Unimontes Universidade Estadual de Montes Claros | Secretaria de Desenvolvimento Econômico, Ciência, Tecnologia e Ensino Superior | Minas Gerais Governo de Todos | PIBID Unimontes Programa Institucional de Bolsa de Iniciação à Docência
Apoio: FAPEMIG | FADENOR

ALGORITMO DE EVOLUÇÃO DIFERENCIAL APLICADO AO JOB SHOP SCHEDULING PROBLEM UTILIZANDO RANDOM KEYS

Autores: MATEUS FELLIPE ALVES LOPES, JOÃO BATISTA MENDES, RAMÓN SOUZA SILVA RODRIGUES, VICTOR DE FREITAS ARRUDA, DIEGO LEAL MAIA

Introdução

Este trabalho tem como objetivo apresentar os resultados parciais das pesquisas realizadas através da Iniciação Científica pela Unimontes, tendo como tema a resolução do *Job Shop Scheduling Problem* (JSP), ou Problema de Alocação, através de métodos de algoritmos evolutivos, em especial o algoritmo de Evolução Diferencial (ED), que se baseia no mecanismo de busca do operador de mutação diferencial. Este operador gera novas soluções a partir de vetores diferenciais construídos com pares de soluções candidatas retiradas da própria população. O ED mostra-se como sendo um algoritmo capaz de gerar bons resultados para o JSP, por ter características tais como, robustez, fácil implementação, versatilidade e eficiência, assim como abordado por Tasgetiren (2006).

De acordo com Dasgupta o problema pode ser definido como:

> Existe um número de agentes e um número de tarefas. Podemos alocar qualquer agente a qualquer uma das tarefas, porém, cada alocação tem um custo que pode variar dependendo da tarefa e agente específicos. É necessário que todas as tarefas sejam feitas, designando exatamente um agente para cada tarefa de modo que o custo total (a soma) de todas as alocações seja minimizado. (DASGUPTA et al., 2008, p.2130).

O JSP clássico pode ser apresentado da seguinte maneira: Existem *n jobs*, cada um composto de várias operações que determinam o tempo de execução de cada *job*. Esses *jobs* devem ser processados em *m* máquinas sendo que cada operação utiliza uma das *m* máquinas de acordo com o tempo de duração desse processo. Cada máquina pode processar no máximo uma tarefa por vez. O problema consiste em encontrar um escalonamento das operações que minimize o custo somado de todas as alocações, respeitando as restrições existentes.

O Problema de Alocação é encontrado com frequência em aplicações práticas em várias áreas do conhecimento humano, como em indústrias (planejamento da produção) ou nas instituições de ensino (problema de alocação de salas de aula). Porém, sua característica é de um problema que se enquadra na categoria dos problemas *NP-Difíceis*, ou seja, é difícil encontrar uma solução ótima para o mesmo em tempo hábil (que varia conforme o tamanho do problema) em problemas de larga escala. Um estudo detalhado sobre o problema de alocação pode ser visto em Soares (2011).

Material e métodos

A. Estrutura proposta para o problema

Seja uma população de soluções candidatas, representada por $Xt = \{xt,i; i = 1, ..., N\}$ Image not found or type unknown, em que *t* é o índice da geração corrente e *i* é o índice do indivíduo na população. Cada indivíduo na população corrente é representado por um par de vetores com mesmo tamanho. A aptidão do indivíduo é armazenada no final do primeiro vetor, sendo este a representação da sequência de tarefas em cada máquina, como ilustrado pela figura 1A que apresenta uma solução para o problema de 3 máquinas e 3 tarefas. O atributo *F* corresponde à qualidade do indivíduo (solução), sendo este o tempo necessário para que todas as tarefas sejam executadas em todas as máquinas (*makespan*). Inicialmente este primeiro vetor aloca as tarefas em cada máquina de forma aleatória.

O segundo vetor da estrutura, ilustrado na figura 1B, também é preenchido inicialmente com valores aleatórios (*Random Keys*) limitados entre *-2.0* a *2.0*. De acordo com Bean (1994), a representação por *Random Keys* codifica uma solução com valores aleatórios. Esses valores são usados como chaves de classificação para decodificar a solução, eliminando, assim, o problema de viabilidade usando codificações cromossômicas que representam soluções de uma maneira mais suave.

Desta maneira, as chaves aleatórias foram adaptadas para serem utilizadas no processo de geração de novas populações do algoritmo ED, que em seguida serão ordenadas para modificar as sequencias das tarefas no primeiro vetor de uma solução, tais procedimentos serão descritos durante o processo de esclarecimento do algoritmo.

* Apoio financeiro: FAPEMIG

ISSN: 1806-549X

B. ED adaptado

O mecanismo de busca do algoritmo de evolução diferencial utiliza vetores diferenciais criados a partir de vetores da própria população. Dois indivíduos são aleatoriamente selecionados da população corrente, originando um vetor com a diferença entre estes dois vetores. Este vetor, por sua vez, é somado a um terceiro indivíduo, também escolhido aleatoriamente, produzindo então uma nova solução mutante através de uma perturbação da diferença entre dois indivíduos. Esse processo pode ser ilustrado pela seguinte formulação:

$$V(q+1) = X?(q) + ?(X?(q) - X?(q))$$

Onde *X?, X?, X?*são os vetores escolhidos aleatoriamente e distintos entre si. Em que a diferença é multiplicada por um *? >* 0 da geração *q*, sendo denotada por diferença ponderada. Este processo resulta o vetor doador *V (q+1)*.

Usando este procedimento, obtém-se uma população mutante de cerca de 70% do tamanho da população corrente. Até este momento utilizamos apenas os vetores de chaves aleatórias dos indivíduos para montagem da população mutante.

O próximo passo do algoritmo é o cruzamento, os indivíduos (vetores com chaves aleatórias) da população corrente *Xt* são recombinados com os indivíduos da população mutante que são escolhidos aleatoriamente, produzindo a descendência ou população de soluções teste *Ut*. A recombinação utilizada é do tipo discreta com probabilidade *C* ? [0, 1], assim como descrevida em Guimarães (2009) para a versão clássica do ED. O parâmetro *C*, controla a fração de valores do vetor que são copiados do vetor mutante, foi definido com o valor de *0.8* para o algoritmo proposto. Isto demonstra que o algoritmo se comporta melhor com a maior chance de que a solução teste contenha muitos valores herdados do vetor mutante.

Após o cruzamento tem-se dois vetores de chaves aleatórias para um vetor de alocação de máquinas e tarefas, as chaves aleatórias geradas através do cruzamento serão agora utilizadas para a criação dos vetores de alocação da população filha para o cálculo das aptidões da população. Nessa etapa os vetores gerados pelo cruzamento são ordenados, é garantido anteriormente que não possuam valores iguais no vetor, e comparados com suas antigas posições antes do ordenamento. Será gerado um novo vetor de alocação em que cada valor deste novo vetor será correspondente a posição da sua chave aleatória após a ordenação.

Finalmente o valor da função objetivo é avaliado para as novas soluções geradas pelo cálculo do *makespan*. Em seguida, cada solução teste *ut,i* é comparada com seu correspondente na população corrente, no caso *xt,i*. Se a solução teste é melhor do que a solução corrente *xt,i*, a solução corrente é eliminada e seu lugar passa a ser ocupado por *ut,i*. Caso contrário, a solução teste é descartada e a solução corrente sobrevive, permanecendo na população da próxima geração. O processo se repete até que uma condição de parada seja satisfeita. Neste trabalho, usou-se o número de gerações como critério de parada.

Resultados e discussão

Após a implementação, em Python 2.7, do algoritmo proposto, obtivemos resultados parciais para problemas testes gerados aleatoriamente, tais resultados podem ser observados nos gráficos da figura 2. Para cada problema teste o gráfico demonstra a média de 20 interações do algoritmo. Os gráficos são compostos por duas linhas, a linha verde representa o melhor valor de *fitness* (aptidão) para cada geração e a linha azul representa a média dos *fitness* de todas as soluções em cada geração.

Os resultados encontrados foram satisfatórios, pois foi possível melhorar os tempos de *makespan* dos problemas, assim como em problemas grandes com 20 tarefas e 15 máquinas. Sendo assim, pode ser observado que houve êxito na implementação do algoritmo proposto, pois os objetivos foram alcançados.

A próxima etapa do presente trabalho consiste no aperfeiçoamento do algoritmo com introdução de técnicas de busca local objetivando aprimorar a qualidade das soluções geradas. Em seguida utilizaremos alguns problemas de *benckmarks* acerca de problemas de alocação de recursos existentes no algoritmo, com a finalidade de haver uma melhor comparação com outros métodos para resolução do JPS.

* Apoio financeiro: FAPEMIG

ISSN: 1806-549X

Considerações finais

O desenvolvimento do presente trabalho possibilitou uma análise mais profunda sobre o JPS e de suas características, utilizando métodos de algoritmos evolutivos para resolução do mesmo. Além disso, também permitiu uma pesquisa bibliográfica acerca do algoritmo ED, sendo possível a elaboração e implementação de uma adaptação do algoritmo para otimização dos resultados para o JPS.

Agradecimentos

Esse trabalho foi financiado pela Fundação de Amparo à Pesquisa do Estado de Minas Gerais (FAPEMIG) pelo Programa Bolsa a Iniciação Cientifica e Tecnológica Institucional (PIBIC).

Referências bibliográficas

BEAN, J.C. **Genetic Algorithms and Random Keys for Sequencing and Optimization**, ORSA journal on computing. vol. 6, n.2, p. 154-160, 1994.

DASGUPTA, D. et al**. A Comparison of Multiobjective Evolutionary Algorithms with Informed Initialization and Kuhn-Munkres Algorithm for The Sailor Assignment Problem**. In: PROCEEDINGS OF THE 2008 GECCO CONFERENCE COMPANION ON GENETIC AND EVOLUTIONARY COMPUTATION, 2008, New York, NY, USA: ACM, 2008, p. 2129–2134.

SOARES, H. C. A**. Um estudo sobre o Problema de Alocaçã**o. 2011.Monografia (Ciência da Computação), Universidade Federal de São Paulo, São Paulo.

STORN, R. M; PRICE, K. V. **Differential evolution - a simple and efficient adaptive scheme for global optimization over continuous spaces**, Technical Report TR-95-012, International Computer Science Institute (ICSI), University of California, Berkeley, 1995.

TASGETIREN, M. F. et al. **A Particle Swarm Optimization and Differential Evolution Algorithms for Job Shop Scheduling Problem.** vol. 3, n. 2, p. 120-135, 2006.

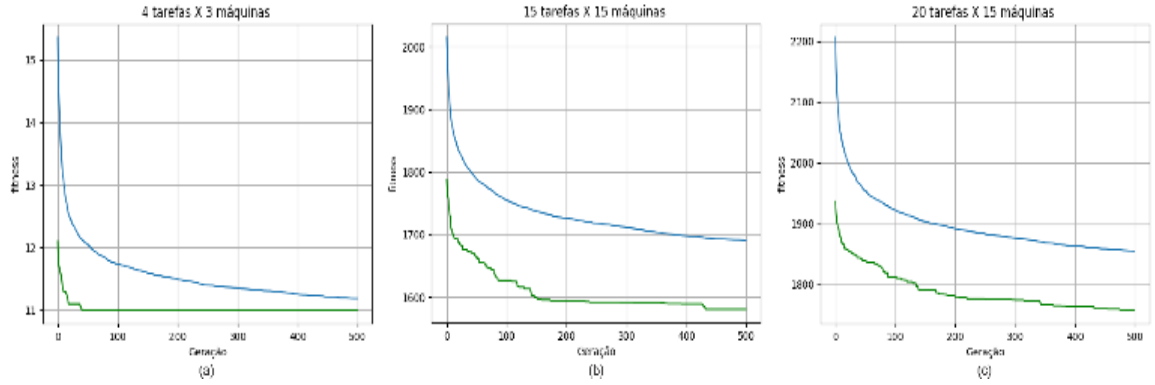
Máquina 01			Máquina 02			Máquina 03			
1	3	2	3	2	1	3	1	2	*F*

(a)

1.87	-0.62	0.42	-0.29	-1.32	1.53	-1.96	-1.01	1.29

(b)

Figura 1. Estrutura proposta para o indivíduo para o problema de JSP.



* Apoio financeiro: FAPEMIG

ISSN: 1806-549X

Figura 2. Gráficos gerado pelo algoritmo dos resultados obtidos para 3 problemas.

* Apoio financeiro: FAPEMIG