Resolvendo Job Shop Scheduling Através de Algoritmo Genético

Dalisson Carlos Almeida Figueiredo

Abstract—O presente trabalho tem por objetivo implementar uma solução para o Job Shop Scheduling utilizando algoritmo genético, como aprendido durante as aulas de computação evolucionário. Esta implementação em matlab segue as recomendações recebidas através do enunciado do problema.

Index Terms—Job Shop Scheduling, Algoritmo Genético

1 Introdução

Considerado um problema NP-Completo combinatorial, Job Shop Scheduling consiste em um problema de otimização no qual tarefas são designadas a recursos em determinada sequência com o objetivo de maximizar o output de deterrminado sistema [2]. Em sua versão mais básica n tarefas de tempo de completude variável são designadas a m máquinas com tempo por tarefa variável, sob a suposição de que duas máquinas não podem trabalhar em uma mesma tarefa ao mesmo, uma máquina também não trabalha em mais de uma tarefa por vez e todas as tarefas tem a mesma ordem predefinida entre as máquinas, o objetivo é minimizar o tempo total gasto (makespan) pelas m máquinas para completar todas as n tarefas.

Por se tratar de um problema prático importante no mundo real, vários pesquisadores trabalharam em diversas soluções para resolvê-lo sob diferentes circunstâncias com diferentes suposições [3]. No presente trabalho é apresentada uma implementação de solução desenvolvida para a versão mais simples do problema utilizando algoritmo genético, detalhes da implementação como operadores de mutação e cruzamento serão discutidos ao longo das seções posteriores deste relatório. A seguir faz-se a introdução dos métodos básicos que constituem esse trabalho.

1.1 Versão do Job Shop Scheduling Implementada

Seguindo o comando do trabalho prático, o problema implementado trata-se de uma das mais simples versões do problema. Cada tarefa passa por uma sequencia obrigatória de máquinas, cada máquina executa uma tarefa por vez e cada máquina executa apenas um único tipo de tarefa.

1.2 Algoritmo Genético

Introduzido por Holland em 1975, o algoritmo genético consiste em uma heurística estocástica que usa métodos semi-aleatórios de pesquisa para optimizar soluções. Seu processo de funcionamento é baseado na teoria evolucionária e seu método de funcionamento trabalha com uma população de soluções cuja alteração, seja ela através de de cruzamento entre outras soluções ou através de perturbações por mutação, ao longo de certa quantidade de gerações, aplicando aqui o conceito de adaptação (quão boa uma solução é) e sobrevivência do mais adaptado (melhores soluções, em geral, tem maior chance de produzir descendentes ou serem selecionadas para a próxima geração), produzem soluções melhores e de fitness (grau de adaptação) superiores ao encontrados na população inicial.

Cada solução é representada através de genes existindo diversas técnicas de codificação de soluções, bem como técnicas de seleção populacional e mutação. A seguir discutimos as técnicas empregadas neste trabalho

2 Desenvolvimento

O presente trabalho foi desenvolvido com base em diversos artigos, sobretudo em um trabalho publicado por um grupo de pesquisadores chineses [3] que demonstra a efetividade do algoritmo genético na solução do problema em questão.

2.1 Representação Genética de Soluções

Uma das etapas mais importantes da implementação do algoritmo genético é a escolha da representação do problema. A representação deve permitir representar de forma efetiva cada indivíduo da população como solução para o problema em questão. Em virtude da simplicidade do problema foi escolhida uma forma simplificada da implementação no trabalho desenvolvido por (Ye, LI e Yan, CHEN, 2010). Cada individuo é representado por um vetor cuja sequencia de números representa o a sequencia de tarefas nas máquinas, isto é, seja uma instância de problema com sete tarefas, uma representação de um indivíduo pode ser dado por [1,7,4,2,3,5,6], onde a tarefa um entrará primeiro na primeira máquina, seguido da tarefa sete, quatro, dois e sua sequência. A representação não só possibilita representar a solução como se mostra eficiente no cálculo do makespan.

2.2 Função de Fitness

A função de fitness definida deve ser escolhida de acordo com as características do problema. Por se tratar de um problema onde o fitness máximo não é conhecido, foi escolhida a função 1/makespan, assim, quanto mais tempo demore uma instância a executar maior seu makespan e menor será seu fitness em consequência. Auxiliando a escolha de soluções optou-se por implementar a técnica de janelamento (windowing), como discutido em sala de aula é uma técnica utilizada para corrigir

os problemas de perda de pressão na seleção, o que ocorre quando o fitness as soluções são muito próximos (caso do problema em questão) e deslocamento da função fitness. O uso de janelamento tornou o algoritmo, mais estável e efetivo na busca por melhores soluções.

2.3 Escolha de Soluções para Reprodução

A técnica de escolha para soluções candidatas a serem pai foi feita através da roleta proporcional ao fitness, que se mostrou efetiva em produzir bons resultados. A roleta foi implementada para escolher cada um dos pais que participaram de algum cruzamento.

2.4 Operador de Cruzamento e Taxa de Cruzamento

O operador de cruzamento deve ser funcional para o tipo de problema a ser enfrentado, nesse caso, trata-se de um problema combinatório quando a ordem dos números no vetor codifica a solução, seguindo esses requisitos, optou-se pela função Order CrossOver (também conhecido como C1) [1]. Operador introduzido por Dave Davis em 1985. O operar consiste em cruzar duas soluções primeiro recortando um pedaço diretamente de um dos pais e o reproduzindo diretamente em mesma posição sobre o filho, em seguida o restante do genoma do filho é retirado do segundo pai e reproduzido seguindo a ordem em que aparece neste. Seguindo a implementação dos pesquisadores chineses [3] a taxa de reprodução foi mantida a 60% de possiblidade de cruzamento.

2.5 Operador de Mutação e Taxa de Mutação

No desenvolvimento deste trabalho foi implementado a mutação por swap simples entre duas posições do genoma. Simplesmente se escolhe duas posições dentro da solução e as troca de posição, novamente seguindo os pesquisadores chineses a taxa de mutação foi mantida 0.2.

2.6 Modelo de População eTamanho

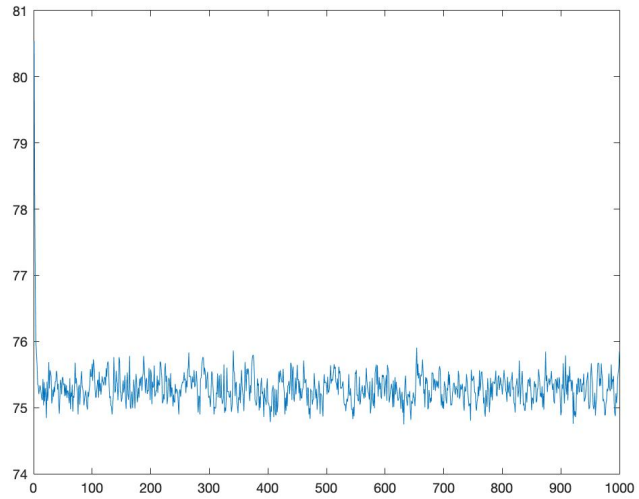
O modelo de população escolhida foi o modelo geracional, onde cada solução sobrevive por unicamente uma geração e toda a população é substituída pela geração seguinte. O tamanho da população foi mantido em trezentos indivíduos, número determinado experimentalmente e suficiente para demonstrar a convergência em todas as instâncias disponibilizadas junto à especificação do problema. Além disso o número de gerações foi mantido em mil, suficiente para atingir resultados que demonstrem a efetividade do algoritmo genético na solução dos problemas.

3 Resultados

Os gráficos a seguir representam o makespan médio ao longo de mil gerações para todas as instâncias fornecidas no problema. Nota-se a redução clara do makespan ao longo de gerações e a convergência do modelo.

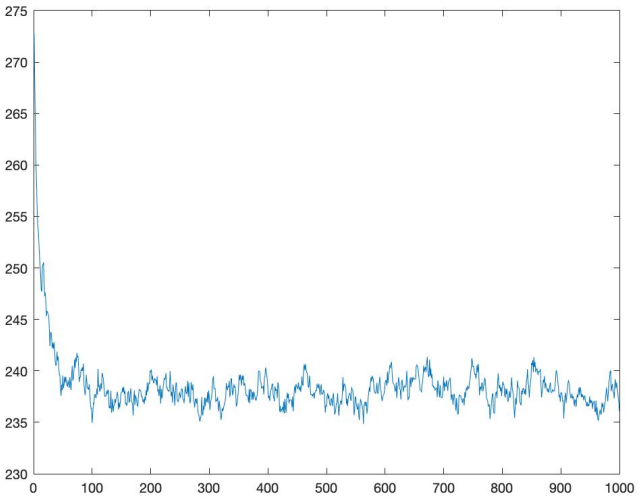
3.1 Três jobs

Fig. 1. Instância de 3 tarefas



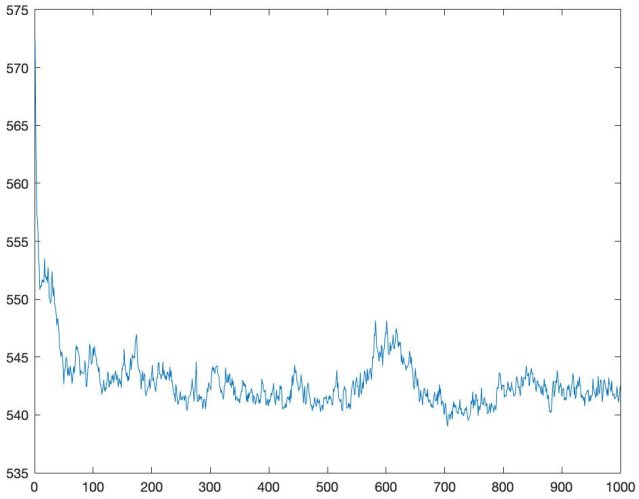
3.2 Dez jobs

Fig. 2. Instância de 10 tarefas



3.3 Vinte e Cinco Jobs

Fig. 3. Instância de 25 tarefas



4 Conclusão

A implementação do trabalho prático possibilitou esclarecimento sobre o conteúdo apresentado em sala de aula, além de ser uma oportunidade interessante para os alunos desenvolverem pesquisa sobre pesquisas e trabalhos implementados na área da computação evolucionária.

Desenvolver este trabalho acrescentou enormemente para a experiência em sala de aula, ao passo que possibilitou não só aplicar os conhecimentos adquiridos em um problema relevante do mundo real, como se tornou uma oportunidade de contato com trabalhos acadêmicos no campo da heurística e optimização

References

[1] Darrell Whitley and Nam-Wook Yoo. Modeling simple genetic algorithms for permutation problems. In Foundations of genetic algorithms, volume 3, pages 163–184. Elsevier, 1995.
[2] Takeshi Yamada and Ryohei Nakano. Job shop scheduling. IEE control Engineering series, pages 134–134, 1997.
[3] LI Ye and CHEN Yan. A genetic algorithm for job-shop scheduling. Journal of software, 5(3):269–274, 2010.