Algorimos Genéticos em Recomendação de Carteira de Ações

LUCIANO PÁDUA SABENÇA *

*Ciência da Computação - Graduação
E-mail: 136700@ic.unicamp.br

***Resumo* – Este trabalho trata da implementação duma aplicação de algoritmos genéticos para recomendação de carteira de ações para investimentos de longo prazo. Fazemos um breve paralelo desse problema com problemas clássicos da literatura, especialmente o "Problema da Mochila" (ou Knapsack Problem). Comparamos duas possíveis funções de fitness e suas implementações usando um dialeto de Lisp, Clojure, além de verificarmos e discutimos também se algoritmos genéticos apresentam bons resultados nesse problema em específico.**

***Palavras-chave* – Algorítimo Genéticos, Mercado de Ações, Clojure**

I. INTRODUÇÃO

O trabalho consiste numa implementação de algoritmo genético, aplicada num problema real: a orientação de investimento em ações. Algoritmos genéticos são muito estudados e aplicados para diversos tipos de problemas, especialmente problemas de otimização combinatório, onde o crescimento exponencial das combinações faz uma busca aleatória inviável já para instâncias relativamente pequenas.[1][2]

O mercado de ações é um mercado bem conhecido pelas oportunidades e também pela sua volatilidade inerente. Apesar disso, o mercado de ações está entre os investimentos mais utilizados, principalmente em países com taxas de juros menores que a do Brasil atualmente. O problema de recomendação de ações para uma carteira de investimentos é um problema bem simples: dado um *budget* limitado e um conjunto de ações, o problema consiste em recomendar um conjunto de ações que devem ser compradas, limitando o valor delas ao valor do budget, e cujo o risco seja minimizado.

Na Seção 2 descreveremos o problema, variantes dele e os dados usados. Na Seção 3 descreveremos a implementação, entrando em alguns detalhes técnico da implementação, escolha da linguagem e outros pontos técnicos. A Seção 4 descreve e comenta os resultados obtidos. Por fim, as conclusões são apresentadas na Seção 5.

II. TRABALHO PROPOSTO

O mercado de ações já causou a ruina e riqueza de diversas pessoas ao longo da sua história recente recente. Há ainda diversos comportamentos possíveis para o investimento, mirando normalmente diferentes prazos. A grosso modo podemos citar 3 comportamentos típicos de investidores: *Day-Trade*, *Swing Trade* e investimentos de longos prazos. O comportamento típico de um investidor em *day-trade* é uma compra e venda (ou venda e compra, dependendo de como ele crê que o preço da ação se comportará) de uma ação, sempre no mesmo dia. Esse tipo de investimento é, basicamente, uma aposta especulativa de alto risco, visando o lucro no curtissímo prazo. Já os *Swing Trade* é a compra e venda de uma ação específica num prazo muito curto, algo em torno de 15 dias. É um investimento de curto-prazo, visando capturar tendências – normalmente frágeis e/ou sazonais – de uma ação. É um comportamento de investimento com risco considerável e visa sempre ações muito voláteis. Já o investimento mais conservador é o investimento a longo prazo. O investidor a longo prazo normalmente mira eventos como a aposentadoria e costuma estar interessado e ações com poucos riscos, pouco voláteis, já que esse tipo de ações tende a manter seu preço ao longo do tempo.

Sabendo de tudo isso, boas métricas são cruciais para bons investimentos no mercado de ações e há vastissima literatura, de diferentes níveis técnicos, sobre o assunto.[3][4]. Além das diversas formulas e teórias advindas da matemática pura, uma das tendências mais fortes nos últimos anos é usar técnicas da inteligência artificial e aprendizado de máquina para a melhorar as recomendações de compra de ações e prever riscos.

Este trabalho sugere aplicar as técnicas de algoritmos genéticos para a recomendação de uma carteira de investimentos em ações com objetivos de longo-prazo. Criamos para isso – e somente com fins acadêmicos – duas possíveis funções de *fitness* e fizemos vários testes visando comparar os diversos parâmetros possíveis para os algoritmos genéticos. Os dados usados foram baseados nos valores da ações das empresas do índice S&P 500 [5] de 2010.

A primeira função de fitness é a baseada na somatória de todas as estabilidades das ações selecionadas:

$$fitness = \sum_{i=1}^{n} estabilidade(x) \tag{1}$$

sendo que a estabilidade de uma ação x é definida como:

$$estabilidade(x) = preço(x) - beta(x) * desvioPadrao(preço(x)) \tag{2}$$

A segunda função de fitness é baseada no conceito de risco de uma ação e é calculada como:

$$fitness = \sum_{i=1}^{n} preco(x) - \sum_{i=1}^{n} risco(x) \tag{3}$$

onde o risco de uma ação x é calculada como:

$$risco(x) = \frac{1}{beta(x) * desvioPadrao(preco(x))} \quad (4)$$

Ambas as formulas usam dois conceitos em comum: beta e desvio-padrão. O beta de uma ação é uma formula clássica para calcular o risco da ação e documentada nas referências[6]. Vale a pena destacar que fizemos uma pequena adaptação no beta para adaptá-lo aos dados. Por último, o desvio-padrão foi calculado em cima do preço de fechamento das ações. Adotamos o desvio-padrão por causa do principio que uma ação com preço estável terá um desvio-padrão baixo no seu preço.

Vale a pena comparar o problema apresentado com o clássico "Problema da Mochila" (Knapsack Problem), um conhecido problema NP-Completo[7]. Nele temos uma mochila com um tamanho máximo, M, e um conjunto de items, cada um com peso p_i e valor v_i, e gostaríamos de maximar o valor dos items a serem carregados, dentro do limite de peso da bolsa. Fazendo uma analogia simples, temos que o budget disponivel seria o equivalente ao peso máximo da mochila M, o que queremos fazer é maximizar o uso do budget ou maximizando a estabilidade (no caso da primeira função de fitness) ou minimizando o risco (no caso da segunda função de fitness).

III. Implementação do Trabalho

O trabalho foi implementado em Clojure[8], um dialeto moderno de LISP que roda sobre a Máquina Virtual Java (JVM). A escolha levou em conta diversos motivos: é uma linguagem altamente dinâmica, possui um *REPL* muito bom, a abordagem funcional permite testes rapidamente, é uma linguagem altamente portável, entre outros motivos. Para auxiliar a visualização e tratamentos básicos dos dados usamos a biblioteca Incanter[9].

Escolhemos por implementar um *namespace* genérico que implementa as primitivas básicas de um algoritmo genético, como, por exemplo, os operadores genéticos de mutação e crossover, além do algoritmo de seleção. Optamos por escolher um modelo de seleção baseado no método da roleta para selecionar os indivíduos mais aptos na geração da nova população.

Usando esse namespace genérico, implementamos as funções que fazem o mapeamento do cromossomo para o resultado e também as funções de fitness. Definimos uma lista de ações, l, e a usamos para fazer a função de mapeamento. A função de mapeamento somente verifica se o bit n está setado, se estiver adicionamos a ação $l[n]$ à lista de ações a serem adicionadas na carteira de ações. A implementação do cromossomo é feita através de um array de inteiros, onde cada bit do número inteiro é mapeado para um bit do cromossomo.

IV. Resultados

Para análise do algoritmo e dos parâmetros fizemos vários testes para determinar o quanto um parâmetro influência no fitness encontrado. **Em todos os testes mantivermos sempre o parâmetro de budget – ou seja, o valor máximo a ser gasto – em 1000 doláres.** Como já foi dito, criamos duas funções de fitness, uma baseada no conceito de "estabilidade", já definido acima e a outra baseada no conceito de "risco", também já definida acima.

Tabela I
Resultados - Fitness baseado em Estabilidade

# População	Taxa Mutação	Taxa Crossover	Gerações	Fitness
100	0,2	0,7	100	987,667
100	0,2	0,9	100	989.177
100	0,5	0,7	100	985.879
100	0,8	0,7	100	982.507
100	0,2	0,7	300	989.403
100	0.2	0,9	300	990.478
300	0,2	0,7	100	989.215
300	0,2	0,7	300	991.266

Tabela II
Resultados - Fitness baseado em Risco

# População	Taxa Mutação	Taxa Crossover	Gerações	Fitness
100	0,2	0,7	100	949.915
100	0,2	0,9	100	934.157
100	0,5	0,7	100	926.249
100	0,8	0,7	100	900.411
100	0,2	0,7	300	946.174
100	0.2	0,9	300	958.169
300	0,2	0,7	100	968.100
300	0,2	0,7	300	969.720

Várias informações interessantes podem ser obtidas através dos dados das tabelas. A primeira é que, para ambos os casos, a medida que a taxa de mutação aumentava, o resultado do fitness tendia a piorar, como fica claro olhando-se as linhas 3 e 4 das tabelas. Esse comportamente é totalmente esperado, considerando-se que a mutação é um operador aleatório, quanto maior sua taxa, mais aleatória fica a tendência de busca no espaço de soluções possíveis e, portanto, tem-se que a resposta tende a se beneficiar menos das características de hereditariedade de um algoritmo genético e tende a piorar.

Uma segunda observação relevante – e também esperada – é que quanto maior a quantidade de gerações, melhor tende a ser a resposta, apesar de maior o tempo de processamento requerido. O motivo disso é claro: quanto maior quantidade de gerações, mais soluções foram checadas no espaço de soluções possíveis e, obviamente, maior a disposição para encontrar resultados melhores.

Por último, vale ressaltar que o tamanho da população é um importante fator para a qualidade da solução. A tendência é que quanto maior a população, melhor a resposta encontrada. Os motivos para isso são os mesmos para o parâmetro de quantidade de gerações: quanto maior a população, mais soluções são checadas e, portanto, maior a tendência de se melhorar a resposta. De novo, aumentar o tamanho da população aumenta o tempo requirido para terminar o processamento.

Todavia, há diferenças entre os dados das duas tabelas. A maior é que, enquanto para I aumentar para 0.9 a taxa de crossover melhora a solução, para II piora a solução, pelo menos para a mesma quantidade de população e de gerações (linha 2, das tabelas). Isso pode acontecer por vários motivos, mas citaremos somente um: a aleatoriedade das soluções geradas durante o segundo teste gerou um minimo local, dificil de ser superador, e como a taxa de croossover diminuíu a aleatóriedade das solução, ficamos "presos" nesse minimo local.

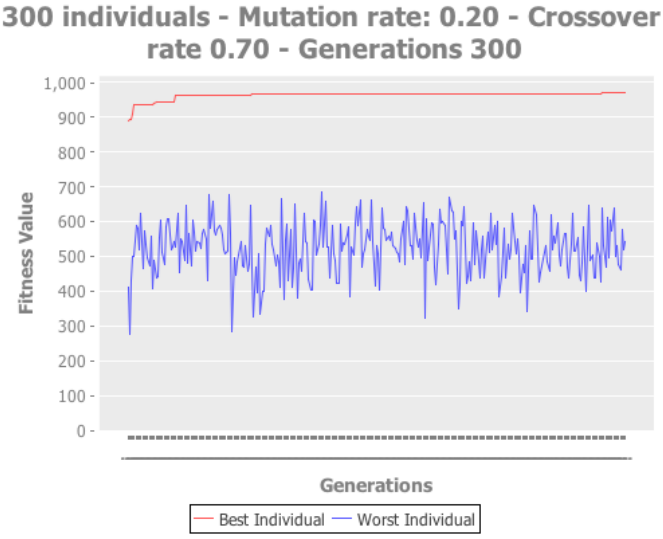

Figura 1. Gráfico Fitness Baseado em Risco

Gerando os gráficos de comparação entre o melhor e o pior indivíduo de cada geração, obtemos informações muito semelhantes, por causa disso, só colocaremos um gráfico nesse relatório, da função de "fitness baseado em riscos", linha 7 de II.

Analisando o gráfico, podemos ter *insights* básicos: o problema converge rapidamente para uma solução boa, não ótima, porém, uma vez encontrada essa solução, sair dela para uma melhor é dificil e toma tempo. O segundo insight que podemos ter é que a há uma tendência de se melhorar a solução ruim com o passar do tempo. Essa tendência porém é difusa e possui recaídas, claramente visivéis no gráfico com as quedas no fitness dos piores indivíduos com o passar das gerações.

V. Conclusões

Algorítmos genéticos estão ganhando muita força nos últimos tempos e aplicar esse paradigma em problemas do mundo real pode trazer benefícios e também danos, se o problema ou a solução não for bem modelada. Acreditamos que esse trabalho mostrou bons resultados na aplicação de algoritmos genéticos para um problema prático e importante, provando que algoritmos genéticos podem ser usados na resolução de problemas no mercado de ações.

O principal ponto fraco do trabalho se refere às funções de fitness usadas, elas não são adequadas para orientação efetiva no investimentos, isso porque elas não são baseadas em nenhum estudo concreto sobre o tema e nem em dados externos às ações, como o tamanho da dívida e faturamento da empresa. Isso, porém, não atrapalha o objetivo principal do trabalho: aplicar algoritmos genéticos num problema real e analisar sua efetividade

Do ponto de vista de implementação, o trabalho mostra que Clojure se mostrou uma escolha viável e prática para esse tipo de projeto.

+————————-+

Referências

[1] Z. Michalewicz, *Genetic Algorithms + Data Structures = Evolution Programs (3rd Ed.)*. London, UK, UK: Springer-Verlag, 1996. 1

[2] R. LINDEN, *Algoritmos Genéticos (2a edição)*. BRASPORT. [Online]. Available: https://books.google.com.br/books?id=it0kv6UsEMEC 1

[3] N. Teebagy, *The Math Behind Wall Street: How the Market Works and how to Make it Work for You*. Four Walls Eight Windows, 1998. [Online]. Available: https://books.google.com.br/books?id=CfNsHQAACAAJ 1

[4] J. Paulos, *A Mathematician Plays the Stock Market*. Basic Books, 2004. [Online]. Available: https://books.google.com.br/books?id=FUGI7KDTkTUC 1

[5] S&P. (2016) S&p 500. [Online]. Available: http://us.spindices.com/indices/equity/sp-500 1

[6] Investopedia. (2016) Beta information. [Online]. Available: http://www.investopedia.com/terms/b/beta.asp/http://www.investopedia.com/articles/stocks/04/113004.asp 2

[7] R. M. Karp, *Complexity of Computer Computations: Proceedings of a symposium on the Complexity of Computer Computations, held March 20–22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York, and sponsored by the Office of Naval Research, Mathematics Program, IBM World Trade Corporation, and the IBM Research Mathematical Sciences Department*. Boston, MA: Springer US, 1972, ch. Reducibility among Combinatorial Problems, pp. 85–103. [Online]. Available: http://dx.doi.org/10.1007/978-1-4684-2001-2_9 2

[8] Clojure. (2016) Clojure. [Online]. Available: https://clojure.org/ 2

[9] Incanter. (2016) Incanter. [Online]. Available: http://incanter.org/ 2