Aplicação de Algoritmo PSO no Problema do Caixeiro Viajante

CCO-727 Otimização Inteligente de Sistemas Produtivos – Profº Edilson Kato

Diego Luiz Cavalca – diego.cavalca@dc.ufscar.br

**RESUMO**

O presente trabalho visa analisar e comparar, em termos de desempenho computacional, a qualidade das soluções obtidas usando Algoritmos de Núvens de Partícula (do inglês, PSO) para solução do Problema do Caixeiro Viajante (PCV). O PCV foi o primeiro problema apresentado na literatura com o intuito de buscar um melhoramento nas rotas de veículos, no entanto trata-se de um problema NP-hard, ou seja, não é possível encontrar uma solução ótima em tempo computacional válido. Visto que soluções ótimas não são ideais para esse tipo de problema, métodos heurísticos são estudados para buscar resultados satisfatórios.

Particle Swarm Optimization (PSO) ou Otimização por Nuvem de Partículas é uma metaheurística que surgiu da intenção de simular o comportamento de um conjunto de pássaros em vôo, com seu movimento localmente aleatório, mas globalmente determinado. Esta técnica tem sido muito utilizada na resolução de problemas contínuos não-lineares e pouco explorada em problemas discretos. Este artigo apresenta o funcionamento desta metaheurística, com novas adaptações, para sua aplicação em problemas de otimização discreta. Ao final, são apresentados resultados de experimentos computacionais para algumas instâncias do Problema do Caixeiro Viajante (PCV), disponibilizadas na TSPLIB, a fim de demonstrar a eficiência do método na resolução de problemas desta categoria. O melhor resultado conhecido para a base de dados TSPLib EIL51.tsp foi 426. O algoritmos proposto nesse trabalho atingiu como melhor resultado o valor de XXXX para o dado problema.

1. **INTRODUÇÃO**

O Problema do Caixeiro Viajante (PCV) consiste em estabelecer uma rota que passe por cada nó de um grafo, uma única vez, retornando ao nó inicial no final do percurso.

Este roteiro Hamiltoniano deve ser feito de modo que a distância total percorrida seja mínima. O conjunto de rotas possíveis para o PCV Simétrico, isto é, se a distância do ponto a ao ponto *b* é igual ao do ponto *b* ao ponto *a*, é *a*, é o resultado de todas as combinações possíveis e pode ser calculado por *(n − 1)!*, sendo *n* o número de nós.

Este problema pertence a classe de problemas conhecida por NP-Hard, isto é, não existem algoritmos com limitação polinomial capazes de resolvê-lo. Assim a quantidade de passos de um algoritmo que possa solucioná-lo otimamente não pode ser dada por uma função polinomial do tamanho de sua entrada. Logo, apenas os problemas de pequeno porte podem ser solucionados de forma ótima.

Problemas maiores tornam-se tornam-se inviáveis através dos métodos exatos, haja vista o esforço computacional que seria exigido para resolvê-los. Muitas abordagens de algoritmos heurísticos, que fornecem soluções factíveis próximas da ótima, têm sido desenvolvidas para resolver os problemas NP-Hard, apresentando soluções aproximadas e as algumas vezes ótimas para o problema.

Diante disto, Particle Swarm Optimization (PSO) ou Otimização por Nuvem de Partículas é uma metaheurística que surgiu da intenção de simular o comportamento de um conjunto de pássaros em vôo, com seu movimento localmente aleatório, mas globalmente determinado.

1. **APLICAÇÃO DO ALGORITMO PSO PARA O PROBLEMA DO CAIXEIRO VIAJANTE**
   1. **ALGORITMO *PARTICLE SWARM OPTIMIZATION***

Conforme GOLBARG, os pesquisadores naturais vêm examinando o comportamento de grandes grupos de animais, como pássaros, peixes e mamíferos ao longo do tempo. Este comportamento apresenta padrões de coordenação síncrona de movimentos, o que viabilizam a fuga de predadores e facilitam o deslocamento dos componentes do conglomerado. Assim, este enxame, ou nuvem, de componentes desempenha um importante papel na vida destas espécies, garantindo que elas evoluam nos diversos contextos que as permeiam.

Inpirado nos trabalhos da biologia, a abordagem computacional que mimetiza esse comportamento surgiu do trabalho combinado do biólogo Frank Heppner, do psicólogo James Kannedy e do engenheiro eletricista Russel Eberhart. Consiste em mimetizar o comportamento adotado em multidões de indivíduos, totalmente inspirado na biologia, os quais formam a base para esta meta-heurística.

Nestes sistemas é assumido que os componentes são indivíduos autônomos, sendo que a auto-organização surge da aplicação de princípios que comuns aos componentes do bando. Segundo GOLDBARG, alguns princípios mais comum neste tipo de associação são:

* Os componentes do bando são autônomos;
* As regras e o tipo de coordenação são simples;
* O número de componentes do bando é grande;
* Existe um objetivo compartilhado pelos componentes;
* As interações entre os agentes são de natureza local;
* Existe flexibilidade e robustez;
* A auto organização é emergente e não é processada de forma centralizada.

Em PSO a população é chamada nuvem. Uma nuvem é um número de partículas que se movem em um espaço n-dimensional, dentro de um subespaço de busca S (VESTERS- TRØM; RIGET, 2002).

Cada partícula p, numa dada iteração t, tem uma posição em Rn, X(t) e uma velocidade de deslocamento nesse espaço, V(t). Possui também uma memória contendo

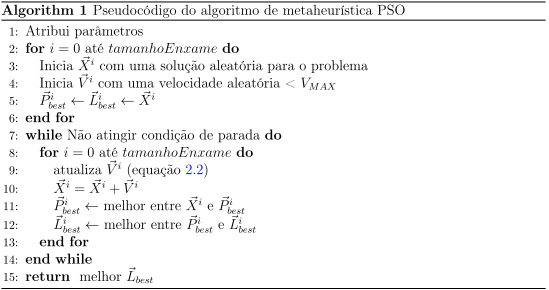
sua melhor posição já alcançada, Pbest, é a melhor posição já alcançada pelas partículas vizinhas a p, Lbest, que é determinada pela topologia de vizinhança implementada. É importante ressaltar que X(t), V(t), Pbest, Lbest, são vetores n-dimensionais, sendo n determinado pelo problema que está sendo atacado pelo algoritmo. No problema que este artigo aborda, o problema do Caixeiro Viajante, n seria o número total de cidades que devem ser visitadas como apresentado em Wang et al. (2003).

A posição de cada partícula representa uma solução potencial para o problema.

O objetivo do algoritmo é movimentar essas partículas a fim de fazer com que elas se tornem soluções ótimas para o problema (TORÁCIO, 2008). Para essa movimentação, uma partícula tem três opções chamadas neste trabalho de M1, M2, M3:

* Seguir seu próprio caminho (M1);
* Seguir em direção a sua melhor posição já encontrada (M2);
* Seguir em direção à melhor posição da vizinhança (M3).

Toda a movimentação das partículas ocorre de acordo com a qualidade das mesmas, como a posição de uma partícula equivale a uma solução em potencial para o problema, o objetivo do algoritmo é guiar a partícula pelo espaço de busca rumo às posições de melhor qualidade já conhecidas em busca de soluções consideradas ótimas para o problema. Esta qualidade, ou nível de aptidão, é calculada através de uma função de avaliação (fitness), responsável por informar o quão boa é uma determinada posição de uma partícula num dado instante.

**Quadro 1** – *Pseudocódigo* Algoritmo

**Fonte:** .

Conforme se pode observar no algoritmo 1, pseudocódigo do PSO, a nuvem é iniciada no tempo *t* = 0, espalhando-se as partículas aleatoriamente no espaço S. Para cada partícula, suas posições são iniciadas (X(0) = Pbest(0) = Lbest(0)) juntamente com suas velocidades V (0). Feito isso, inicia-se o processo iterativo. A posição de cada partícula é alterada num determinado tempo *t* adicionando a velocidade à posição atual da partícula de acordo com a equação:

X(t+1) = X(t) + V(t+1)

A velocidade da partícula numa determinada iteração t é baseada na melhor posição já alcançada pela partícula (Pbest(t)) e pela melhor posição alcançada pelos vizinhos. Como apresentado anteriormente, a definição da melhor solução da vizinhança (Lbest(t)) depende da topologia de vizinhança implementada, assunto da próxima subseção. A função de atualização da velocidade, no tempo t+1, é definida por:

V(t+1) = w\*V(t) + c1\*φ1\*(Pbest(t) – X(t)) + c2\*φ2\*(Lbest(t) – X(t))

Onde w é um coeficiente chamado na literatura de inércia da partícula. Ela determina o quanto a velocidade anterior influencia na velocidade atual, equivalendo a autoconfiança da partícula (TORÁCIO, 2008). Um alto valor para w faz com que a partícula procure seguir mais o seu próprio caminho ao invés de optar por seguir as melhores posições já alcançadas por ela mesma e por suas vizinhas.

Já os coeficientes c1, c2, φ1 e φ2 determinam a influencia de Pbest(t) e Lbest (t) sobre a partícula p. Onde φ1 e φ2 são coeficientes aleatórios que variam de 0 a 1, enquanto c1 e c2 são pré-configurados e influenciam o quanto a partícula vai tender a seguir cada uma das duas opções. Caso c1 seja significativamente maior que os coeficientes c2 e w, a partícula tenderá a ir de encontro à posição do Pbest (t), por outro lado, caso c2 seja o coeficiente de maior valor, ela tenderá à posição do Lbest (t).

Para que a velocidade não fique muito alta fazendo com que a partícula se disperse demasiadamente do restante da nuvem, pode ser definido um parâmetro de velocidade máxima (VMAX) que limita o valor da velocidade da partícula. Após a atualização da velocidade e da posição de todas as partículas o processo é repetido nas próximas iterações ate o final da execução do algoritmo.

============\*\*\*\*\*==================\*\*\*\*\*============

Vale ressaltar que esta meta-heurística foi proposta inicialmente para otimizar funções não lineares contínuas. Entretanto, há diversas pesquisas que investigam maneiras de aplicar este algoritmo para problemas discretos, dentre eles o de Clerc (2004) aplicado ao Problema do Caixeiro Viajante, o qual serviu de base para a implementação desenvolvida neste artigo.

Todavia, as modificações propostas para o PSO aplicadas a problemas discretos podem ser ditas de certa forma, elegantes, pois elas preservaram toda a estrutura do algoritmo PSO original e, além disso, insere o PSO discreto em uma nova classe de problemas. As mudanças sugeridas foram: Os vetores de posição atual e melhor posição foram substituídos por valores discretos, e o vetor velocidade, por probabilidades onde um valor discreto em particular, modificará uma determinada iteração.

1. **METODOLOGIA**

Os algoritmos foram desenvolvidos utilizando a ferramenta MATLAB 2015a, testados sob o sistema operacional OSX El Captain 10.11.6, numa máquina com processador de 2.26GHz Intel Core 2 Duo, memória RAM de 8gb 1067MHz DDR3 e placa de vídeos NVIDIA GeForce 9400 256MB e 1Tb de HD.

Os parâmetros para configuração dos algoritmos utilizados nos testes se diferenciaram para cada Algoritmo Genético desenvolvido, de acordo com suas especificações, tentando manter uma métrica similar a fim de obter um resultado comparativo de ambas as abordagens ao final do trabalho.

* 1. **DEFINIÇÕES E REGRAS DO PCV**

O PCV não permite rotas com valores duplicados, ou seja, cada cidade deve ser visitada apenas uma vez e a rota deve ser encerrada na cidade de partida inicial.

* 1. **PROJETO DE ALGORITMOS PROPOSTOS**
     1. **ALGORITMO CLÁSSICO**

O operador de cruzamento PMX faz a combinação dos cromossomos sem permitir que genes iguais apareçam na mesma prole, portanto, foi o usado na aplicação.

O cruzamento é feito assim que os pais são selecionados, estes são passados como parâmetro para o método *crossover PMX* que retorna os filhos resultantes da combinação de genes. Esse processo é realizado até preencher toda a nova população que deve ter o mesmo tamanho da população de progenitores.

A mutação não é um operador fundamental para o algoritmo, mas seu uso pode ser fundamental para se alcançar bons resultados. Os indivíduos da aplicação têm 3% de chance de sofrerem mutação, e esta aplicada foi por inversão.

* + 1. **ALGORITMO SEM *CROSSOVER***

Para este AG, foram realizados testes com populações de 20, 100, 200 e 400 indivíduos. Cada população foi submetida a 100, 500, 1000, 3000, 5000 e 10000 gerações possíveis.

Em tempo, para este algoritmo, não serão necessários outros parâmetros, uma vez que, conforme já citado, este realiza a operação de mutação em larga escala, abrangendo sob todo a população, em todas as gerações.

1. **RESULTADOS**

Para a avaliação do PCV os dados usados foram os do problema EIL51.tsp, disponibilizado pela TSPLIB (TSPLIB, 2010), que possui tamanho de 51 nós. Os resultados obtidos variavam de acordo com mudanças na quantidade de gerações e no tamanho da população de indivíduos. Custos menores puderam ser alcançados com o aumento de gerações e de indivíduos das populações, entretanto, por conta desses acréscimos o tempo de execução da aplicação tornou-se mais longo.

* 1. **AG CLÁSSICO**

O melhor resultado alcançando pelo algoritmo foi **442.281**, atingindo um resultado abaixo de 10% em comparação com o melhor resultado obtido na literatura para este conjunto de dados, atingindo assim à meta proposta. Este resultado foi alcançado com uma população de **10000** indivíduos e um limite máximo definido de **200** gerações; o resultado em si foi encontrado na geração número **125**.

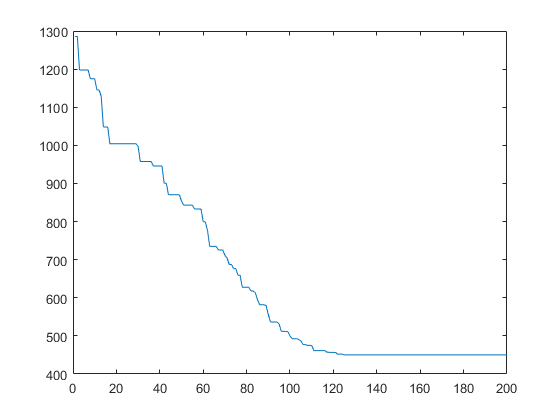
A tabela 1 mostra um resumo geral dos resultados do AG clássico.



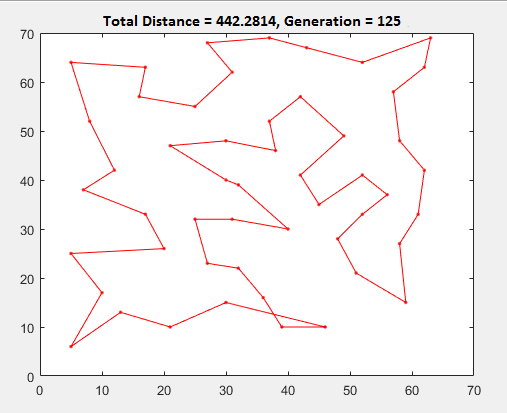
**Tabela 1:** *Resumo geral de resultados do AG clássico.*

A taxa de mutação foi definida em 3%. Se esta taxa for muito alta as características boas de alguns filhos gerados no cruzamento podem ser perdidas, o que não é o objetivo do método, por isso o ideal para o algoritmo é manter uma taxa menor do que 5%. Menores custos puderam ser alcançados com o aumento de gerações e de cromossomos das populações. No entanto, quando as gerações chegavam à faixa de 150 até 200, a população não sofria evolução significativa, tomando isto como concluiu-se que, gerações com número superior a 200 não eram interessantes para o estudo.

A seguir, as figuras 1 e 2 mostram o comportamento deste resultado em relação as gerações:

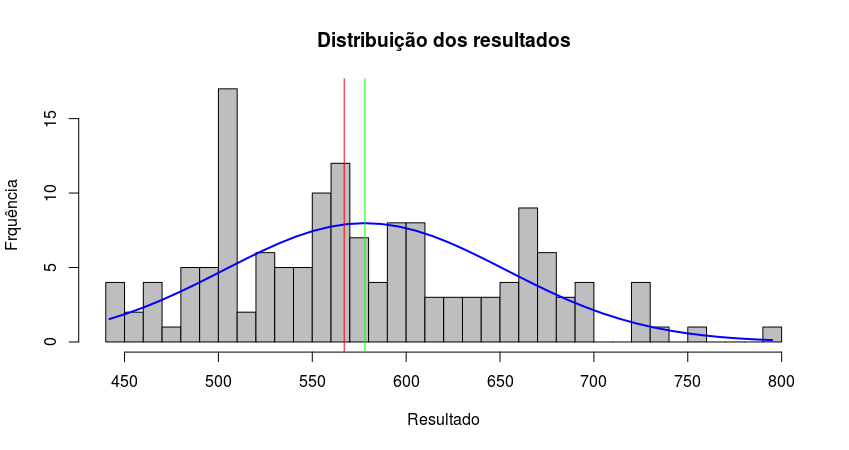


**Figura 1:**Relação entre fitness (caminho) e gerações do melhor resultado.



**Figura 2:** Representação do melhor caminho encontrado pelo AG clássico

Abaixo, a figura 3 mostra a distribuição dos resultados obtidos nos testes do AG:



**Figura 3 -** *Distribuição dos resultados do AG*

Portanto, fica evidente que o comportamento geral deste AG é estável, com pouca diferença entre a média (linha verde) e a mediana (vermelha) para todos os casos, assumindo assim uma distribuição normal dos resultados.

* 1. **AG SEM *CROSSOVER***

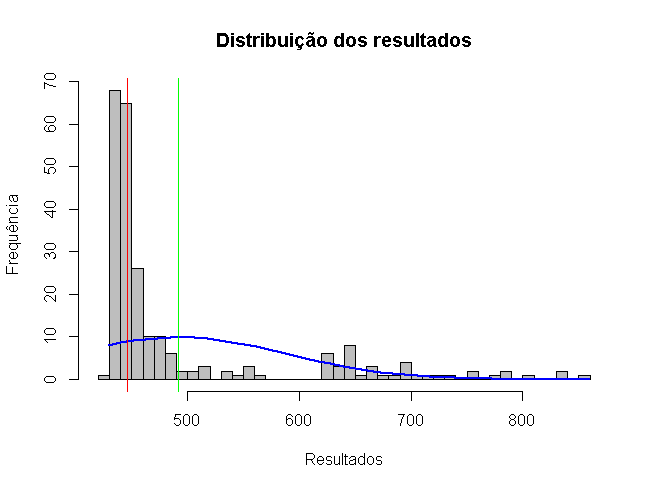
Conforme o resumo geral sobre a performance deste AG apresentado abaixo, (reportado de forma completa no ANEXO I), o melhor resultado obtido para o PCV aplicado ao conjunto de dados EIL51.tsp foi **428.98**:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| *Min.* | *1st Qu.* | *Median* | *Mean* | *3rd Qu.* | *Max* |
| 428.98 | 439.0 | 445.6 | 492.1 | 477.3 | 860.0 |

**Tabela 2:** *Resumo geral de resultados do AG sem crossover.*

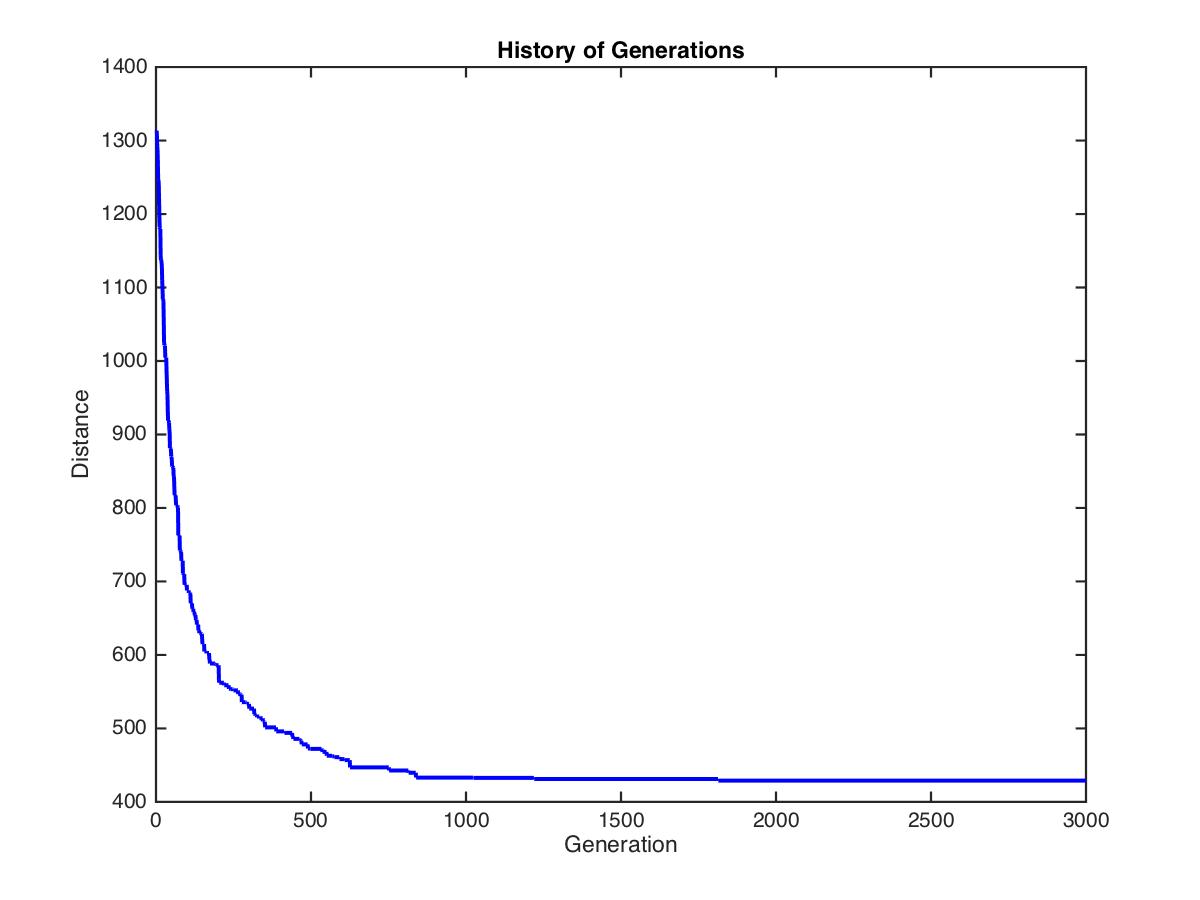
É interessante notar que este resultado está a apenas **2.98** pontos do melhor resultado conhecido para este conjunto de dados. Além disso, a variância observada teve valor de 96.3, mostrando uma consistência no desempenho geral deste AG.

Também vale ressaltar que a mediana dos resultados obtido situou em **445.6**, atingindo um resultado abaixo de 10% em comparação à meta, conforme pode ser vista representada pela linha vertical vermelha no gráfico de distribuição dos resultados abaixo:



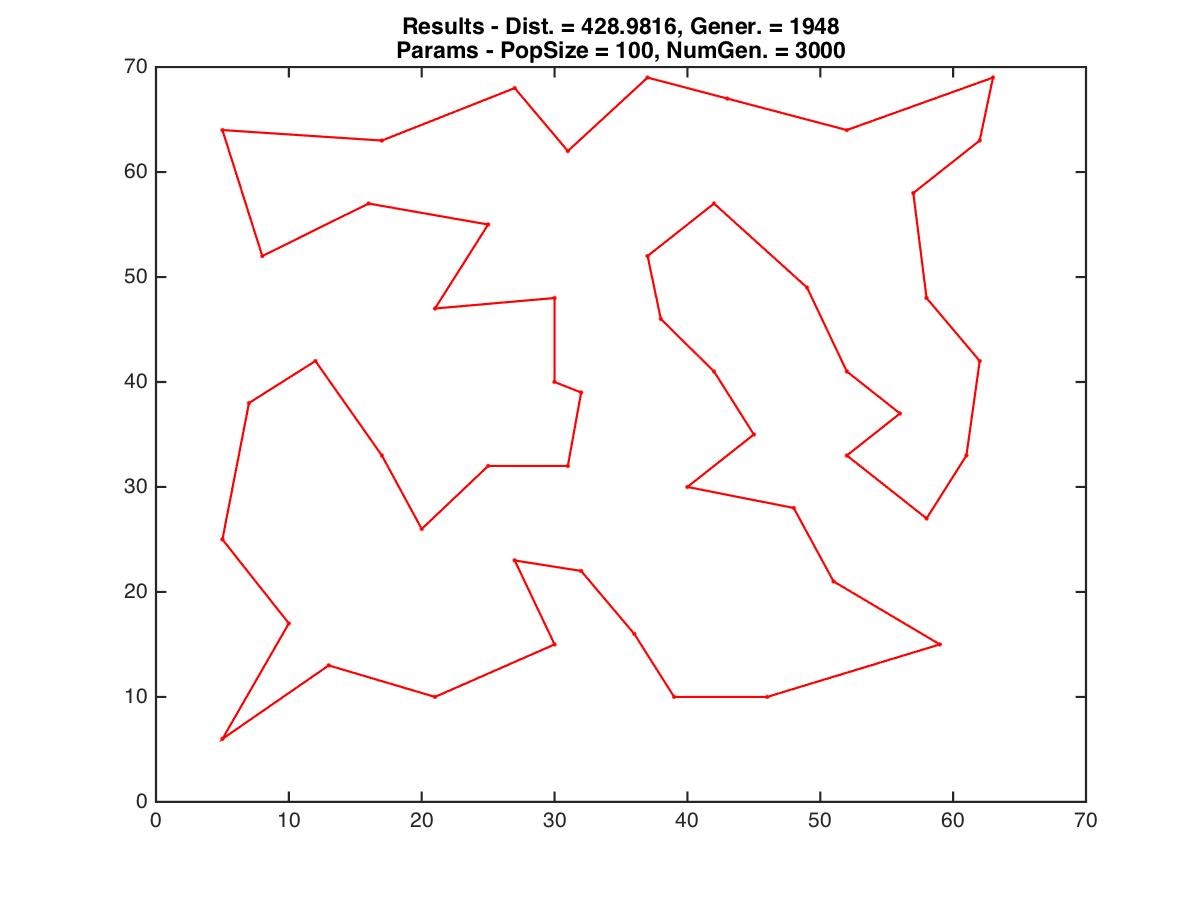
**Figura 4 -** Distribuição dos resultados do AG

Vale ressaltar que o melhor resultado deste AG foi encontrado com o número populacional de **100** indivíduos e um limite máximo definido de **3000** gerações; o resultado em si foi encontrado na geração número **1948**. O comportamento deste resultado em relação as gerações pode ser visualizado abaixo:



**Figura 5:**Relação entre fitness (caminho) e gerações do melhor resultado.

Em tempo, a distribuição visual do caminho encontrado pode ser visualizada a seguir:



**Figura 6:**Representação do melhor caminho encontrado pelo AG.

Portanto, o AG sem *crossover* se mostra eficiente para o PCV, obtendo um resultado extremamente próximo ao melhor encontrado, ainda que testado em sob uma pequena quantidade de configuração de parâmetros.

Dada a característica complexa do PCV, este resultado corrobora a tese de que ao remover a operação de cruzamento do AG obtém-se uma maior variabilidade populacional, apoiando a aproximação a rota de menor custo a cada geração, se mostrando mais eficiente quando comparado ao AG clássico, conforme abordado neste trabalho.

1. **CONCLUSÃO**

As regras que estabelecem o tamanho da população e da utilização da operação de *crossover* são extremamente importantes para o desenvolvimento do AG. Esta definição é atingida de maneira natural usando uma população de tamanho considerável, de acordo com o problema estudado.

Portanto, pode ser deduzido que o cruzamento é mais efetivo quando aplicado em grandes populações. Os resultados deste estudo indicam esta tese, mesmo este sendo limitado a um problema em particular de programação genética.

É também interessante notar que embora as melhorias de desempenho rápido ocorrem com um tamanho da população menor, uma população maior ajuda o algoritmo genético encontrar melhores soluções. Isto é causado pela acumulação mais lenta de estatísticas mais exatas quando se utiliza a população maior. É evidente que a rápida acumulação de estatísticas precisas permitiria que o melhor dos dois mundos.

Além disso, fica evidente que diante da característica complexa do PCV, com restrições quanto a composição do cromossomo, é válido analisar diferentes estratégias na implementação do AG a fim de lidar com a diversidade da população entre as gerações, o que leva assumir mecanismos híbridos nas operações do algoritmo genético, como foi proposto neste trabalho, ao abordar um AG sem operação de *crossover*, no qual garante a diversidade da população através da mutação em larga escala dos indivíduos a cada geração.

Como próximos trabalhos, podem ser analisados os tempos computacionais envolvidos, a fim de mensurar o esforço estrutural nas várias simulações distintas. No AG sem *crossover*, podem ser exploradas novas formas de mutação, incluindo a própria diminuição do número de mutações realizadas. Novos estudos podem ser direcionados a fim de explorar a correlação entre as variáveis envolvidas nas configurações de cada AG proposto no trabalho, assim definindo com exatidão quais parâmetros, e em quais escalas, afetam diretamente os resultados de maneira eficiente.

**REFERÊNCIAS**

ARTERO, A. O. **Inteligência artificial teórica e prática.** São Paulo: São Paulo, 2008.

de Garis, H. (1990a). Genetic Programming: Building Nanobrains with Genetically Programmed Neural Network Modules, Proceedings of the International Joint Conference on Neural Networks, San Diego, CA, Junho 1990

Benevides, P. F., Konowalenko, F., Costa, D. M. B., Nunes, L. F., & Barboza, A. O. (2011). Aplicação de Algori itmos Genéticos e Simulated Annealing para o Problema do Caixeiro Viajante em uma Situação Real de Distribuição de Produtos. In *I Congresso Brasileiro de Engenharia de Produção*.

COPPIN, B. Algoritmos genéticos. In: \_\_\_\_\_\_. **Inteligência artificial.** Rio de Janeiro, 2013. cap. 14, p. 334-360.

HOLLAND, J. H. **Adaptation in natural and atificial system.** An Arbor. Universidade of Michingan Press, 1975.

NÉIA, S. S.; ARTERO, A. O.; CANTÃO, L. A. P.; CUNHA, C. B. Roteamento de veículos utilizando otimização por colônia de formigas e algoritmo genético. In: LOPES, et al. (Eds.), **Meta-heurísticas em pesquisa operacional**. cap. 14, p. 219-236, 2013.

Senaratna, N. I. (2005). Genetic Algorithms: The Crossover-Mutation Debate. *Bachelor of Computer Science(Special) of the University of Colombo*, 1–22. Retrieved from http://www.geocities.ws/nis\_nisco/docs/GA.pdf

Spears, W. M., & Anand, V. (2015). A Study of Crossover Operators in Genetic Programming. *Statewide Agricultural Land Use Baseline 2015*, *1*. <http://doi.org/10.1017/CBO9781107415324.004>

TSPLIB - Traveling Salesman Problem Library. Disponível em: <http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/tsp/> Acesso em 30 ago. 2016.

**ANEXO I - ALGORITMOS DESENVOLVIDOS E DEMAIS INSUMOS**

O código-fonte, resultados, análise dos dados (linguagem R), e demais arquivos de cada AG desenvolvido neste trabalho estão disponíveis em <https://github.com/diegocavalca/mestrado/tree/master/CCO-727/Trabalho1-AG_TSP>

A fim de organização e objetividade, cada AG esta disponível nos seguintes endereços:

1. **AG clássico:** <https://github.com/diegocavalca/mestrado/tree/master/CCO-727/Trabalho1-AG_TSP/AG1>
2. **AG sem *crossover*:** <https://github.com/diegocavalca/mestrado/tree/master/CCO-727/Trabalho1-AG_TSP/AG2>