AVALIAÇÃO E REFINAMENTO DE UM ALGORITMO GENÉTICO PARA PROBLEMAS DE OTIMIZAÇÃO MULTICOMPONENTE

Josué Barbosa Ávalos
Universidade Federal de Viçosa
Florestal, MG
josue.barbosa@ufv.br

Marcus Henrique Soares Mendes
Universidade Federal de Viçosa
Florestal, MG
marcus.mendes@ufv.br

RESUMO

Recentemente um novo problema foi proposto para melhor contemplar a interdependencia existente entre as componentes de problemas reais, o Travelling Thief Problem. Este problema é resultante da junção de dois problemas clássicos da computação, o TSP(Traveling Salesman Problem) e o KP (Knapsack Problem). Por se tratar de um problema multicomponente, a solução ótima dos subproblemas não garante a solução ótima do problema como um todo. Diferentes abordagens surgiram para solucionar esse problema. Os algoritmos propostos geralmente fazem uso de Meta Heurísticas e, devido a isso, possuem um grande número de parâmetros a serem definidos. No entanto escolher esses parâmetros adequadamente não é uma tarefa simples, já que os mesmos afetam diretamente a qualidade das soluções encontradas. Para auxiliar nesse processo algumas ferramentas de configuração automática surgiram e se baseiam em pesquisar no espaço de parâmetros as configurações que melhorem o desempenho dos algoritmos de otimização. Esse trabalho tem por objetivo encontrar os melhores parâmetros para um algoritmo genético utilizado na resolução do Travelling Thief Problem, fazendo para isso o uso dessas ferramentas.

PALAVRAS CHAVE. Travelling Thief Problem, Algoritmo Genético, Configuração Automática.

ABSTRACT

Recently a new problem has been proposed to better contemplate the interdependence between the components of real problems, the Traveling Thief Problem. This problem results from the combination of two classical problems of computation, the TSP (Traveling Salesman Problem) and the KP (Knapsack Problem). Because it is a multicomponent problem, the optimal solution of the sub problems does not guarantee the optimal solution of the problem as a whole. Different approaches have emerged to address this problem. The proposed algorithms generally make use of Meta Heuristics, and because of this they have a large number of parameters to be defined. However choosing these parameters properly is not a simple task, since they directly affect the quality of the solutions found. To assist in this process some automatic configuration tools have arisen and are based on searching in the parameter space the configurations that improve the performance of the optimization algorithms. This work aims to find the best parameters for a genetic algorithm used in the resolution of the Traveling Thief Problem, making use of these tools.

KEYWORDS. Travelling Thief Problem, Genetic Algorithm, Automatic Configuration.

1. Introdução

Muitos problemas clássicos na área da computação foram propostos baseados em uma abstração de problemas do mundo real. Com o intuito de contemplar melhor os problemas reais, um problema chamado Traveling Thief Problem (TTP) foi proposto em Bonyadi et al. [2013]. O TTP é uma junção de dois problemas muito conhecidos na computação que são o Problema da Mochila (KP) e o Problema do Caixeiro Viajante (TSP). Assim como a maioria dos problemas reais, o TTP apresenta subcomponentes que são interdependentes entre si. Dessa forma, a resolução ótima dos subproblemas não garante a solução ótima do problema como um todo, e essa interdependência deve ser considerada ao se desenvolver uma solução.

O TTP é um problema pertencente a classe NP-difícil, sendo assim, não é possível encontrar a solução ótima em tempo polinomial. Faz-se necessário a utilização de meta-heurísticas que possibilitem encontrar soluções aceitáveis, em um tempo computacional viável. Alguns trabalhos foram propostos buscando resolver o problema sequencialmente, ou seja, lidar com uma componente e posteriormente com a outra. Entretanto, os algoritmos que se destacaram na resolução do TTP são os que lidam com as componentes concomitantemente. Muitos dos algoritmos pertencentes a segunda classe são baseados em meta-heurísticas, entre eles o Multicomponent Genetic Algorithm (MCGA) Vieira et al. [2017], que busca solucionar o TTP por meio de um algoritmo genético que trabalha evoluindo tanto a componente KP quanto com a componente TSP. Neste, a representação dos indivíduos é feita utilizando dois vetores, um para cada componente. O primeiro vetor armazena a sequência de cidades a ser percorrida (componente TSP), enquanto o segundo é um vetor binário de tamanho igual ao número total de itens presentes nas cidades. Assim, quando um item deve ser pego o índice correspondente do vetor é setado para um. O MCGA possui dois tipos de cruzamento. Para a componente KP é utilizado um cruzamento *N-point*, onde o vetor que representa essa componente é dividido em n partes e combinado com outro vetor também dividido, formando assim um novo individuo. Já na componente TSP, é utilizado um operador de cruzamento *Order-based* que faz o cruzamento de acordo com uma mascára aleatória de bits, sendo que quando a máscara é um, os genes (posições do vetor) dos pais são copiados para os filhos. Os genes restantes (quando a máscara é zero) de um dos pais são ordenados na mesma ordem que aparecem no outro pai e assim são passados para um dos filhos. O procedimento ocorre também para o outro filho, garantindo que não haja indivíduos com percursos inválidos. O MCGA também possui dois operadores de mutação associados as componentes. Na componente KP é utilizado o *Bit-Flip*, que de acordo com uma probabilidade de mutação inverte ou não o valor da posição posição no vetor, alterando o estado de um determinado item. Na componente TSP o operador *2-Opt* é o responsável por mutar os indivíduos. Este, faz a troca entre duas arestas (caminho entre as cidades) em um percurso, sendo esta dependente de uma probabilidade. Ao se utilizar dos métodos acima, a população de indivíduos dobra, assim é utilizado um método de seleção denominado Torneio, onde os melhores indivíduos são selecionados para compor a nova população, de mesmo tamanho da original. Nesse método são selecionados n índivíduos para comparação e apenas o melhor deles é passado para a próxima geração. O MCGA se utiliza dos métodos já citados e vai evoluindo as soluções até atingir um critério de parada, podendo ser tempo ou número de gerações sem melhora. Quando isso ocorre, a melhor solução obtida é retornada.

Algoritmos baseados em meta-heurísticas geralmente resultam em bons resultados para problemas complexos, no entanto exigem um grande número de parâmetros, sendo que a escolhas destes afetam diretamente a qualidade das soluções encontradas. A escolha adequada desses parâmetros não é uma tarefa simples, visto a importância dessa decisão. Assim sendo, faz-se necessário uma busca nos espaços dos parâmetros de forma a selecionar configurações de parâmetros que farão o algoritmo apresentar um melhor desempenho. Este trabalho busca encontrar melhores configurações de parâmetros para o MCGA, visando uma melhoria na qualidade das soluções já encontradas.

O artigo é dividido da seguinte forma: a seção 2 discute a característica dos problemas de

otimização multicomponente. A seção 3 define o problema TTP, tal como formulado na literatura. Já a seção 4 aborda o tema configuração automática de algoritmos e a sua importância. A seção 5 apresenta a metodologia utilizada nos testes, bem como os resultados obtidos. E, por fim, na seção 6 é apresentada a conclusão.

2. Otimização Multicomponente

Muitos problemas de otimização foram propostos com o passar dos anos por profissionais das mais variadas áreas e muitos desses representam problemas existentes na indústria, sendo assim sua resolução se torna importante. É o caso de problemas como o Travelling Salesman Problem, o Knapsack Problem, o Vehicle Routing Problem, dentre outros. Entretanto, com o passar do tempo, as necessidades das indústrias foram mudando e surgiram problemas com uma característica diferente, os problemas multicomponente. Por exemplo, uma companhia de aviação não está interessada apenas na melhor rota para suas viagens, mas também na melhor alocação de pilotos, no melhor agendamento de manutenções, tudo isso de forma simultânea. Fica claro que cada subproblema afeta o outro, já que por exemplo, uma aeronave pode ser alocada para fazer uma viagem que termine próximo ao seu ponto de manutenção.

De acordo com Bonyadi et al. [2013] muito esforço foi gasto na tentativa de desenvolver métodos mais eficientes de resolução desses problemas isolados, ao passo que um esforço mínimo foi gasto com o desenvolvimento de novas abstrações que representem os problemas do mundo real e que se preocupem com a interdependencia existente entre eles. Essa interdependência deve ser levada em conta para que problemas do mundo real sejam contemplados em sua totalidade. Foi pensando nisso que o TTP foi proposto.

3. O Travelling Thief Problem

Assim como definido em Polyakovskiy et al. [2014], o TTP é definido como tendo um conjunto de cidades N = 1,..., n em que a distância d_{ij} entre cada par de cidades i e j é conhecida, com i, j $\in$ N. Toda cidade i com exceção da primeira possui um conjunto de itens M_i = 1,..., m_i. Cada item k existente em uma cidade i é caracterizado pelo seu valor p_{ik} e peso w_{ik}. A solução candidata deve visitar todas as cidades sem repetir nenhuma e retornar para a cidade de partida. os itens podem ser coletados pelas cidades respeitando-se a restrição de capacidade máxima W da mochila. Uma taxa de aluguel R deve ser paga por cada unidade de tempo que se utiliza para finalizar o percurso. As variáveis υ_{max} e υ_{min} descrevem a velocidade máxima e mínima permitida ao longo do percurso, respectivamente. Já $y_{ik} \in \{0,1\}$, ou seja, é uma variável binária igual a 1 caso o item k fosse coletado na cidade i. W_i é o peso total dos itens coletados quando se deixa a cidade i. Portanto, a função objetivo para um percurso qualquer π = (x_1,..., x_n), $x_i \in$ N e um plano de coleta P = (y_{21},...,y_{nmi}) é definida pela equação 1:

$$Z(\pi, P) = \sum_{i=1}^{n} \sum_{k=1}^{m_i} p_{ik} y_{ik} - R \left(\frac{d_{x_n x_1}}{\upsilon_{max} - \nu W_{x_n}} + \sum_{i=1}^{n-1} \frac{d_{x_i x_{i+1}}}{\upsilon_{max} - \nu W_{x_i}} \right) \quad (1)$$

A variável ν tem seu valor definido como $\nu = \frac{\upsilon_{max} - \upsilon_{min}}{W}$. Em termos gerais, a Equação 1 consiste em fazer a soma dos valores de todos os itens coletados durante o trajeto, subtraindo do mesmo um valor referente à taxa de aluguel. O valor total do aluguel é claramente dependente do tempo gasto para se percorrer o trajeto, sendo influenciado diretamente pelo peso da mochila. O objetivo é maximizar a função $Z(\pi, P)$.

4. Ferramentas de Configuração Automática de Algoritmos

Algoritmos de otimização geralmente possuem um grande número de parâmetros que controlam aspectos importantes no seu funcionamento, os quais são muitas das vezes escolhidos de forma manual e empírica pelos desenvolvedores dos algoritmos. Entretanto por influenciar diretamente no comportamento dos mesmos e, por consequente nas soluções, é fundamental que sua

escolha seja adequada, permitindo que o algoritmo consiga encontrar melhores soluções. Alguns trabalhos mostram que quando é feita uma busca nos espaços de parâmetros as soluções apresentam significativa melhora. López-Ibáñez e Stützle [2010], que usaram o irace para fazer o *tuning*[1] do Multi-objective Ant Colony Optmization (MOACO), um algoritmo multiobjetivo baseado na heurística de colônia de formigas, usado no trabalhho para resolver o bTSP. No trabalho em questão os resultados obtidos sobressaíram ao melhor algoritmo na época. Cáceres et al. [2017], utilizou o irace para minimizar o tempo de execução do código de máquina para vários métodos de pesquisa heurística. Segundo os autores, alguns códigos otimizados apresentaram melhoria de até 40 % do tempo de execução. As ferramentas de configuração automática também podem ser usadas para outros propósitos como projetar algoritmos mais complexos, encontrando uma configuração de algoritmos básicos que devem ser usados para compor um algoritmo mais geral.

Neste trabalho o framework utilizado foi o irace. A escolha se deu principalmente pelo fato do mesmo ser bem utilizado e apresentar uma boa documentação.

4.1. Irace

O irace é um framework para configuração automática de algoritmos de otimização, ou seja, busca encontrar automaticamente configurações para os valores de parâmetros de forma a fazer com que o algoritmo apresente um melhor desempenho. Para isso o irace segue o fluxo de execução mostrado na Figura 1, onde cada parte é detalhada em sequência:

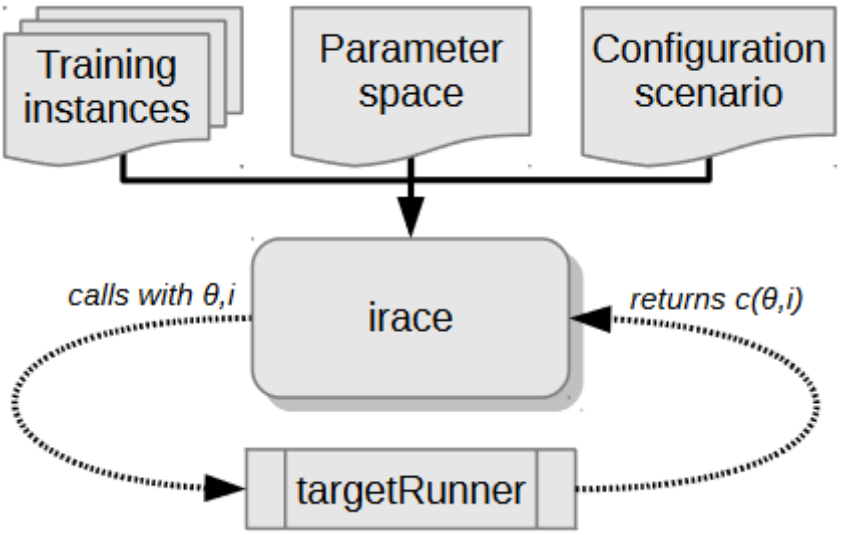

Figura 1: Fluxo de Execução no irace
Fonte: López-Ibáñez et al. [2016b]

- Instância de Treinamento: São as instâncias do problema que serão utilizadas para avaliar a qualidade das configurações criadas.
- Espaço de Parâmetros: É um arquivo que define os possíveis valores que os parâmetros podem assumir. Utilizando esse arquivo, o irace irá criar as configurações a serem testadas nas instâncias.
- Cenário de configuração: É um arquivo que define um conjunto de opções disponíveis no irace, como número de iterações, número de execuções por iterações, diretório de execução, dentre outras opções disponíveis em López-Ibáñez et al. [2016a].

[1] Processo de otimização dos parâmetros de um algoritmo

- Execução Alvo: É um programa que faz a interface entre o algoritmo que se deseja fazer o *tuning* e o Irace. Esse programa é responsável por fazer a passagem dos parâmetros das configurações para o algoritmo, e repassar o resultado do algoritmo para o irace.

Através das instâncias e das configurações criadas, o irace faz a chamada ao algoritmo utilizando o Target Runner e recebe as avaliações do algoritmo alvo. Assim, configurações que apresentam bons resultados são mantidas no processo de otimização, ao passo que as piores configurações são eliminadas.

5. Metodologia e Resultados

Para fazer o *tuning* do algoritmo foram selecionadas um subconjunto de 36 instâncias selecionadas por Polyakovskiy et al. [2014]. Nesse subconjunto, existem 3 números de cidades diferentes, sendo n entre 195 e 3038, dois números de itens por cidade (m), e 2 fatores de capacidade da mochila C $\in$ (3,7). Para cada instância existem ainda três variações (t): quando os valores dos itens não estão relacionados ao peso (*uncorrelated - unc*), qunado os valores não estão relacionados aos pesos porém os itens tem pesos com valores próximos (*uncorrelated with similar weights - uws*), ou quando os valores dos itens estão fortemente relacionados ao seus pesos (*bounded strongly correlated - bsc*).

Essas instâncias foram então divididas em 6 grupos, onde cada um era dado pelo número de cidades da instância e pelo número de itens em cada cidade. A partir de então, no arquivo de espaço de parâmetros foram definidos os intervalos de valores que cada parâmetro do MCGA poderia assumir. Os experimentos foram colocados em execução, sendo que para cada conjunto foram executados 750 vezes, tendo como critérios de parada 8 minutos ou 8000 gerações sem melhora. Essa execução levou cerca de 100 horas para cada conjunto (750 vezes X 8 minutos/vez). Foram obtidas 6 configurações, apresentadas na tabela 1, que posteriormente foram aplicadas ao MCGA. Para fins de comparação, o MCGA foi executado 30 vezes para cada uma das 36 instâncias com as configurações obtidas no irace, sendo a média usada para comparação. Para execução dos experimentos, foi utilizado um servidor da UFV, tendo como processador Intel Xeon E5-2430 v2, 6GB de RAM e executando o sistema Linux Ubuntu 16.04.

Os resultados obtidos foram então comparados com os já existentes em Vieira et al. [2017]. Neste trabalho foram utilizados 200 indivíduos como tamanho de população, ambos torneios das etapas de seleção com tamanho 2, todos números de soluções elites iguais a 12, três pontos de corte para o operador de cruzamento e 0.2% de probabilidade de mutação. A comparação é mostrada na Tabela 2, na qual os melhores resultados são destacados em negrito, juntamente com o desvio padrão (σ) após o *tuning*.

Tabela 1: Configurações obtidas após o processo de *tuning*

	195		**783**		**3038**	
	582	1940	2346	7820	9111	30370
Número de Indivíduos	150	100	100	100	100	100
Tamanho de Torneio	3	4	4	4	4	4
Tamanho de Torneio na etapa de Cruzamento	3	3	3	3	3	4
Número de Soluções elites	9	9	14	13	8	14
Número de soluções elite (cruzamento)	9	11	13	14	7	9
Número de soluções elite (mutação)	9	10	8	8	8	14
Número de pontos de corte	2	3	2	2	3	3
Probabilidade de mutação	0.0066	0.0014	0.0031	0.0029	0.0099	0.0096

Tabela 2: Comparação entre os resultados obtidos após fazer o tuning do MCGA

n	m	t	C	MCGA	MCGA Tuning	(σ) após tuning
195	582	bsc	3	84385.5	**84703**	2174.65
			7	117426	**119887**	3169.68
		unc	3	60017.3	**61102**	1332.03
			7	75303.3	**76263**	1071.6
		usw	3	30871.8	**31051**	737.552
			7	**54357**	53699	1155.11
	1940	bsc	3	229432	**230508**	1917.51
			7	384304	**384430**	2363.32
		unc	3	174002	**174161**	968.2
			7	249938	**250233**	816.777
		usw	3	**115337**	115163.5	505.026
			7	191539	**192985**	2121
783	2346	bsc	3	284904	**285710**	7909.43
			7	481395	**491063**	9003.57
		unc	3	206739	**207221**	2111.78
			7	286227	**286474**	5595.47
		usw	3	139672	**140277**	1772.95
			7	232023	**232650**	3780.99
	7820	bsc	3	**939702**	835765	30039.4
			7	**1483880**	1307280	45581.4
		unc	3	577930	**587233**	57972
			7	**900138**	843808	10252.4
		usw	3	**393142**	284547	42696.7
			7	**702622**	223210	11929
3038	9111	bsc	3	**816356**	814713	4512.5
			7	**1360530**	1302318	49642
		unc	3	58605	**60229**	2193.7
			7	**404174**	395422	7282.11
		usw	3	121877	**125021**	4046.5
			7	375058	**381761**	6329
	30370	bsc	3	**1113600**	1098127	25729.4
			7	2362480	**2373104**	9637.51
		unc	3	**-1531310**	-1582061	39734
			7	**-717599**	-725212	8451.7
		usw	3	-748034	**-740153**	7442.05
			7	**-434909**	-446057	11853

6. Conclusão

Muitos algoritmos para problemas de otimização são configuráveis. No MCGA, por exemplo, temos o tamanho dos torneios, o número de soluções elites, a probabilidade de mutação, o número de cortes na etapa de cruzamento, dentre outros. Assim, a configuração de algoritmos pode ser tratada também como um problema de otimização onde se busca um conjunto de parâmetros que produza o melhor resultado quando aplicado ao algoritmo.

Neste trabalho foi buscada melhorias para o MCGA através de ferramentas de configuração automática de parâmetros. Pode-se observar através dos resultados uma notória melhora na qualidade das soluções (22 em 36). Essa aferição nos mostra como é importante fazer buscas no espaço dos parâmetros por melhores configurações, e como a escolha correta dos mesmos pode afetar o desempenho de um algoritmo de otimização.

Referências

Bonyadi, M. R., Michalewicz, Z., e Barone, L. (2013). The travelling thief problem: The first step in the transition from theoretical problems to realistic problems. In *Evolutionary Computation (CEC), 2013 IEEE Congress on*, p. 1037–1044. IEEE.

Cáceres, L. P., Pagnozzi, F., Franzin, A., e Stützle, T. (2017). Automatic configuration of gcc using irace. In *International Conference on Artificial Evolution (Evolution Artificielle)*, p. 202–216. Springer.

López-Ibáñez, M., Cáceres, L. P., Dubois-Lacoste, J., Stützle, T., e Birattari, M. (2016a). The irace package: User guide. *IRIDIA, Université Libre de Bruxelles, Belgium, Tech. Rep. TR/IRIDIA/2016-004*.

López-Ibáñez, M., Dubois-Lacoste, J., Cáceres, L. P., Birattari, M., e Stützle, T. (2016b). The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, 3:43–58.

López-Ibáñez, M. e Stützle, T. (2010). Automatic configuration of multi-objective aco algorithms. In *International Conference on Swarm Intelligence*, p. 95–106. Springer.

Polyakovskiy, S., Bonyadi, M. R., Wagner, M., Michalewicz, Z., e Neumann, F. (2014). A comprehensive benchmark set and heuristics for the traveling thief problem. In *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation*, p. 477–484. ACM.

Vieira, D. K., Soares, G. L., Vasconcelos, J. A., e Mendes, M. H. (2017). A genetic algorithm for multi-component optimization problems: The case of the travelling thief problem. In *European Conference on Evolutionary Computation in Combinatorial Optimization*, p. 18–29. Springer.