

CONFIGURAÇÃO AUTOMÁTICA DE ALGORITMOS HEURÍSTICOS DE OTIMIZAÇÃO ¹

Clara dos Santos Becker ², Marcelo de Souza ³

¹Vinculado ao projeto “Projeto Automático de Algoritmos”.

²Acadêmico (a) do Curso de Engenharia de Software – CEA VI – Bolsista PROIP/UDESC.

³Orientador(a), Departamento de Engenharia de Software – CEA VI – marcelo.desouza@udesc.br.

1. Introdução

Os algoritmos heurísticos são uma abordagem comumente explorada na solução de problemas de otimização combinatória, quando sua solução exata é inviável na prática devido ao elevado tempo de processamento. O desempenho desses algoritmos está diretamente relacionado a boas escolhas dos componentes e estratégias algorítmicos a serem usados, e bons valores para seus parâmetros. Esse processo de escolha, chamado de configuração do algoritmo, pode ser realizado de forma experimental e intuitiva, baseada em tentativa e erro, recombinao componentes e valores na busca das melhores combinações. Tal abordagem está associada a um grande esforço humano e é suscetível a erros. Além disso, esse processo pode demandar muito tempo explorando componentes e valores de parâmetros ineficientes, enquanto boas estratégias podem permanecer desconhecidas devido à alta dimensionalidade do espaço de configurações a ser explorado.

Uma alternativa a esse cenário é o uso de ferramentas, chamadas configuradores, que automatizam o processo de configuração desses algoritmos. Um exemplo de configurador de algoritmos é o irace (López-Ibáñez et al., 2016), que explora modelos de amostragem para gerar configurações, avaliando-as através de *racing*. Configurações com desempenho inferior são descartadas ao longo desse processo, e aquelas de melhor desempenho são usadas para ajustar os modelos de amostragem para a próxima iteração. Técnicas de configuração automática foram largamente exploradas na literatura para produzir algoritmos com desempenho otimizado para problemas de decisão (e.g., KhudaBukhsh et al., 2016) e problemas de otimização (e.g., López-Ibáñez e Stützle, 2012).

Este trabalho apresenta um estudo da configuração automática de algoritmos em dois cenários distintos usando o configurador irace. No primeiro cenário, é configurada a metaheurística GRASP (*Greedy Randomized Adaptive Search Procedure*) para o problema da diversidade máxima (Kuby, 1987). No segundo cenário, o algoritmo guloso iterado é configurado para resolver o problema das p-medianas capacitado (Koskosidis e Powell, 1992). O estudo demonstra os benefícios da configuração automática de algoritmos heurísticos nesses cenários.

2. Abordagem proposta

O problema da diversidade máquina consiste em selecionar um subconjunto de m elementos a partir de um conjunto de n elementos, de forma a maximizar a diferença entre os elementos selecionados. A diferença entre dois elementos é definida conforme a aplicação do problema, utilizando alguma métrica de distância apropriada, como a distância Euclidiana a partir de um conjunto de características. Já o problema das p-medianas considera um conjunto de n pontos e consiste em determinar a localização de p centros, conhecidos como medianas, de modo

que a distância total de cada ponto à sua mediana mais próxima. Na versão capacitada, os pontos possuem demandas a serem atendidas, enquanto as medianas possuem capacidades de atendimento. Além da localização das medianas, deve ser feita a alocação de cada ponto a alguma mediana com capacidade de atendimento disponível.

Para o estudo dos algoritmos para o problema da diversidade máxima, foram exploradas instâncias da literatura. Para o problema das p -medianas capacitado, foi desenvolvido um estudo de caso para o planejamento urbano do campus da Udesc Alto Vale. O campus foi mapeado para um conjunto de pontos, cada qual com sua demanda, e deseja-se alocar recursos, como bebedouros ou lixeiras.

A metaheurística GRASP, proposta por Feo e Rezende (1995), está descrita pelo pseudocódigo à esquerda na Figura 1. O algoritmo guloso iterado (IG – *iterated greedy*), proposto por Ruiz e Stützle (2007), está descrito pelo pseudocódigo à direita na Figura 1. Ambos algoritmos foram implementados na linguagem Python, para a solução dos problemas supracitados.

Figura 1. Pseudocódigo dos algoritmos propostos.

1 procedimento GRASP (M, α, β)	1 procedimento Iterated Greedy (I, α, D)
2 $S_{best} \leftarrow \emptyset$	2 $S \leftarrow$ SoluçãoAleatória()
3 para $i = 1$ até M faça	3 $S^* \leftarrow S$
4 $S \leftarrow$ Construção(α, β)	4 para $i \leftarrow 1$ até M faça
5 $S \leftarrow$ BuscaLocal(S)	5 $S' \leftarrow$ Destrução (S, d_1, d_2)
6 se S é melhor que S_{best} então	6 $S' \leftarrow$ Construção (S', α, β)
7 $S_{best} \leftarrow S$	7 se busca local está ativada então
8 fim se	8 $S' \leftarrow$ BuscaLocal(S')
9 fim para	9 fim se
10 retorna S_{best}	10 se S é melhor que S^* então
11	11 $S^* \leftarrow S$
12	12 fim se
13	13 $S \leftarrow$ Aceita(S', S^*)
14	14 fim para
19	15 retorna S^*

O algoritmo GRASP realiza duas etapas principais. A primeira consiste na construção de uma solução usando uma heurística construtiva semi-gulosa (linha 4). Em seguida, essa solução passa por um procedimento de refinamento usando alguma heurística de busca local (linha 5). Portanto, a cada iteração do algoritmo uma solução é gerada e melhorada. A solução incumbente é atualizada sempre que uma solução melhor é produzida (linha 6 e 7). A solução incumbente é retornada ao esgotar o número total de iterações pré-estabelecido. Os parâmetros da heurística construtiva α e β representam a gulosidade da solução criada. O parâmetro $\alpha \in [0,1]$ representa a gulosidade na escolha do primeiro elemento, já o parâmetro $\beta \in [0,1]$ representa a gulosidade na escolha dos demais elementos da solução. Sendo assim, a heurística construtiva pode ser reduzida a uma construção aleatória quando $\alpha = \beta = 1$, ou a uma construção totalmente gulosa, quando $\alpha = \beta = 0$. Para outros valores, a construção se torna semi-gulosa, com valores mais próximos de 0 tornando o algoritmo mais guloso, e mais próximos de 1 mais aleatório.

O algoritmo IG inicialmente gera uma solução aleatória (linha 1). Durante M iterações, a solução inicial é parcialmente destruída para logo em seguida ser reconstruída (linhas 5 e 6). O tamanho da destruição é dado pelos parâmetros d_1 e d_2 (para definir a quantidade de medianas a remover e de pontos a desalocar, respectivamente). A aleatoriedade da construção é baseada nos

parâmetros α e β (que definem a aleatoriedade na localização de medianas e na alocação de pontos, respectivamente). A busca local pode ser reiniciada de forma opcional caso for identificada estagnação. O algoritmo IG pode ser combinado ao procedimento de busca local para melhorar a solução construída (linhas 7 a 9). Por fim, uma solução é escolhida para a próxima iteração da busca (linha 16).

3. Resultados e discussão

Como pode-se perceber na descrição das metaheurísticas (Figura 1), há várias decisões ao instanciar algum desses algoritmos. Essas decisões envolvem a escolha de componentes algorítmicos, como diferentes heurísticas construtivas ou vizinhanças, bem como valores de parâmetros, como a aleatoriedade da construção, definida pelos parâmetros α e β . Conforme descrito previamente, este trabalho automatiza o processo de construção dos algoritmos propostos ao empregar o configurador irace para encontrar a combinação de componentes e valores de parâmetros que maximizam o desempenho do algoritmo. Os resultados da avaliação dos algoritmos produzidos são apresentados abaixo.

Tabela 1. Desempenho médio do GRASP padrão e sua versão configurada automaticamente, em comparação com uma construção aleatória.

Instância		Construção aleatória	GRASP padrão	Melhoria [%]	GRASP configurado	Melhoria [%]
<i>n</i>	<i>m</i>					
50	5	84,0	115,7	37,7	116,8	39,0
50	5	84,0	116,3	38,5	118,1	40,6
50	15	739,0	750,5	1,6	758,7	2,7
50	15	727,0	739,1	1,7	745,9	2,6
Melhoria média [%]			–	19,8	–	21,2

A Tabela 1 apresenta os resultados do algoritmo GRASP para o problema da diversidade máxima. O GRASP padrão usa uma configuração obtida após experimentos manuais, enquanto o GRASP configurado é aquele cuja configuração foi encontrada pelo irace usando 1000 avaliações (*budget*). Ambas versões são comparadas com uma heurística de construção aleatória repetida (*baseline*). Os três algoritmos usam um total de 200 iterações e os resultados são médias de 10 replicações. Pode-se perceber que o GRASP configurado é ligeiramente superior às outras abordagens, apresentando uma melhoria média de 21,2% na qualidade das soluções, enquanto a configuração padrão melhora em média 19,8% a qualidade das soluções. A versão encontrada pelo irace apresenta construção mais gulosa (menor valor de α), porém com seleção mais aleatória do primeiro elemento (maior valor de β).

A Tabela 2 apresenta os resultados do algoritmo IG para o problema das *p*-medianas capacitado. Para cada instância, é apresentada a solução exata via programação linear inteira (MIP – *Mixed Integer Programming*), usando o solver GLPK com um tempo limite de 1 hora (3600 segundos). Além disso, é apresentado o valor da melhor solução e o valor médio das soluções produzidas pelo algoritmo IG após 10 replicações, bem como o tempo médio de execução. O algoritmo IG foi configurado pelo irace com um total de 300 avaliações e executa 5000 iterações. A Tabela 2 mostra os melhores valores para cada instância em negrito e os valores ótimos (quando conhecidos) são apresentados sublinhados.

Observa-se que a abordagem exata (MIP) conseguiu alcançar a solução ótima em 4 das 10 instâncias no tempo limite de 1 hora de execução. Para as instâncias maiores (com 80 ou mais pontos), o solver retornou apenas soluções viáveis, sem garantir a otimalidade. O algoritmo IG,

ainda que tenha alcançado o valor ótimo em somente 2 das 10 instâncias, conseguiu produzir soluções melhores que o solver nos casos onde a solução ótima não foi alcançada. Além disso, o IG consegue encontrar soluções melhores nessas instâncias em um tempo expressivamente menor.

Tabela 2. Desempenho dos métodos exato e heurístico para o problema das p -medianas capacitado.

Instância		MIP		IG		
pontos	p	valor	tempo [s]	melhor	média	tempo [s]
25	5	1100	0,3	1100	1132,2	29,3
25	7	878	0,6	878	906,9	33,3
49	9	1518	10,3	1532	1579,1	237,7
49	14	1072	5,8	1108	1144,7	331,9
80	16	1880	3600,0	1868	1932,0	1055,6
80	24	1646	3600,0	1529	1562,9	1082,7
90	18	2001	3600,0	1955	2046,3	1306,8
90	27	1797	3600,0	1627	1665,0	1269,3
100	20	2064	3600,0	2106	2156,0	1771,3
100	30	1776	3600,0	1700	1723,2	2013,8

Os resultados apresentados demonstram a capacidade da configuração automática de algoritmos em encontrar boas combinações de componentes e bons valores para parâmetros, melhorando o desempenho de algoritmos e tornando a tarefa de configuração menos onerosa para o pesquisador. Além de contribuir para a solução eficiente dos problemas explorados, este trabalho fornece diretrizes para a aplicação de técnicas de configuração automática para algoritmos de otimização, em particular metaheurísticas.

Palavras-chave: Otimização combinatória. Configuração automática de algoritmos. Metaheurísticas.

Referências

- FEO, T. A.; RESENDE, M. G. C. Greedy Randomized Adaptive Search Procedures. *Journal of Global Optimization*. Kluwer Academic Publishers, 1995.
- KHUDABUKHSH, A. R. et al. SATenstein: Automatically building local search SAT solvers from components. *Artificial Intelligence*, v. 232, p. 20–42, 1 mar. 2016.
- KOSKOSIDIS, Yiannis A.; POWELL, Warren B. Clustering algorithms for consolidation of customer orders into vehicle shipments. *Transportation Research Part B: Methodological*, v. 26, n. 5, p. 365–379, 1992.
- KUBY, Michael J. Programming models for facility dispersion: the p -dispersion and maximum dispersion problems. *Mathematical and Computer Modelling*, v. 10, n. 10, p. 792, 1988.
- LÓPEZ-IBÁÑEZ, M.; STÜTZLE, T. The automatic design of multiobjective ant colony optimization algorithms. *IEEE Transactions on Evolutionary Computation*, v. 16, n. 6, p. 861–875, 2012.
- LÓPEZ-IBÁÑEZ, M. et al. The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, v. 3, p. 43–58, 2016.
- RUIZ, R.; STÜTZLE, T. A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. *European Journal of Operational Research*, v. 177, n. 3, p. 2033–2049, 16 mar. 2007.