

Problema Quadrático de Alocação

Raul S. Silva¹

¹Instituto Metrópole Digital –
Universidade Federal do Rio Grande do Norte (UFRN)
Caixa Postal 1524 – 59078-970 – Natal – RN – Brasil

raul95@lcc.ufrn.br

Abstract. *There are several problems that can be shaped by a general problem. Assigning threads to computational resources and electronic circuit designs are examples of problems that can be modeled by the Quadratic Assignment Problem (QAP), which consists of allocating resources in places, so that the cost of this allocation is minimized. The QAP is a combinatorial optimization problem and is classified as NP-Hard. In this work a state-of-the-art survey of the exact algorithms is presented, then the implementation of an exact algorithm with Branch and Bound is presented. Finally, a comparison of the implementation experiments with the literature is made.*

Resumo. *Existem diversos problemas que podem ser modelados por um problema geral. Atribuir tarefas à recursos computacionais e projetos de circuitos eletrônicos são exemplos de problemas que podem ser modelados pelo Problema Quadrático de Alocação (PQA), que consiste em alocar recursos em locais, de forma que o custo dessa alocação seja o menor possível. O PQA é um problema de otimização combinatória e é classificado como NP-Difícil. Neste trabalho é feito um levantamento do estado da arte dos algoritmos exatos, em seguida é apresentada a implementação de um algoritmo exato com Branch and Bound. Por fim, é feita uma comparação dos experimentos da implementação com a literatura.*

1. Introdução

O Problema Quadrático de Alocação (PQA) é um problema de otimização combinatória que está na classe de problemas NP-Difíceis, uma vez que não é possível exibir um algoritmo polinomial para este problema, a menos que $P = NP$. Tal problema consiste em, dado um conjunto n de locais e um conjunto n de objetos, qual a melhor combinação de alocação desses objetos nesses locais, de forma que a combinação dos custos das distâncias entre os locais com os custos dos fluxos de interações entre esses objetos seja a menor possível.

Em 1957, Koopmans e Beckmann [1] formularam o PQA como um modelo matemático relacionado a atividades econômicas, que estende o Problema Linear de Alocação (PLA), onde além de considerar o custo das distâncias entre os locais, também devem ser considerados os custos de transporte/fluxo entre os objetos que serão alocados. A seguinte equação define matematicamente o problema:

$$\min_{\pi \in S_n} \sum_{i=1}^n \sum_{j=1}^n f_{\pi(i)\pi(j)} d_{ij}$$

Onde π é uma das possíveis alocações de um conjunto S_n que contem todas as permutações dos n objetos nos n locais, $f_{\pi(i)\pi(j)}$ representa o fluxo de interações entre os objetos que estão alocados em i e j , e d_{ij} representa a distância entre os locais i e j . O objetivo é encontrar a permutação que possua o menor custo.

2. Estado da arte

Como o PQA é um problema de otimização combinatória, as soluções propostas na literatura estão divididas em implementações de algoritmos exatos, que conseguem resolver o problema para instâncias de tamanhos limitados em questão de segundos ou horas, e implementações de algoritmos heurísticos, que podem não encontrar a melhor alocação, mas pode encontrar uma que seja próxima a solução ótima em tempo pré-definido. Nessa Seção serão descritos alguns trabalhos encontrados na literatura e apresentados em duas subseções, algoritmos exatos e algoritmos heurísticos.

2.1. Algoritmos exatos

As soluções exatas mais comuns para o PQA usam *Branch and Bound* para observar as possíveis alocações descartando, previamente, as que possuem custos elevados. As soluções não desejadas podem ser calculadas com base em um limite inferior e um limite superior. O cálculo do limite inferior mais conhecido é o limite de Gilmore-Lawler [4 e 6], que consegue fazer um cálculo do limite inferior com complexidade $O(n^3)$ por meio de uma Alocação Linear que recebe as matrizes de fluxos e de distâncias, previamente ordenadas em $O(n^2 \log n)$, e calcula o custo da alocação linear utilizando-se de um algoritmo Hungaro.

Além do cálculo do limite inferior, o *Branch and Bound* pede inicialmente um limite superior, para que ao começar a analisar a árvore das possíveis alocações, possa ser possível definir uma solução inicial e não explorar nós que sejam piores que a solução atual. Além disso, esse limite superior é necessário para que quando uma solução encontrada seja melhor que a solução inicial esse limite possa diminuir e fazer com que menos nós sejam explorados na árvore de alocações.

Para poder achar o limite superior, muitos autores fazem uso de heurísticas para achar uma solução inicial que seja boa e que ajude a diminuir a quantidade de nós explorados. O mais comum é realizar uma busca local na vizinhança de uma solução inicial aleatória, com o objetivo de achar uma solução menor. Em seguida, é feita uma perturbação nessa solução melhorada de forma a realizar outra busca local em uma solução que possua uma vizinhança diferente da atual. Esse processo pode se repetir algumas vezes, e a solução final define o limite superior para o *Branch and Bound*.

Alguns autores como Burkard e Derigs [3], Nugent [7] entre outros, fazem uso de *Branch and Bound* para executar instâncias com tamanho ≤ 20 em algumas horas.

2.2. Algoritmos heurísticos

Para instâncias maiores que 20 objetos requerem um tempo computacional elevado. Para isso alguns autores propõe algoritmos que exibem soluções aproximadas em um tempo pré-definido.

Benlic e Hao [1] fazem o uso de uma busca local para achar uma solução inicial, e em seguida são feitas uma série de perturbações distintas e novas buscas locais são

aplicadas. É um trabalho recente e conseguiu bons resultados para classes de instâncias reais e instâncias geradas aleatoriamente.

Taillard [8], faz busca local em uma vizinhança de uma solução usando uma busca taboo, onde um conjunto de soluções que não são boas, são armazenadas em uma tabela para que não sejam exploradas novamente pelo algoritmo.

3. Implementação

3.1. Técnica de desenvolvimento

No contexto abordado, foi desenvolvido um algoritmo exato com *Branch and Bound* com o limite inferior de Gilmore-Lowler e um limite superior que toma uma solução inicial aleatória e analisa sua vizinhança até que uma solução melhor seja encontrada, e então avaliar a vizinhança dessa nova solução.

3.2. Descrição do algoritmo

A implementação do *Branch and Bound* consiste em receber como entrada a quantidade de objetos a serem alocados, a matriz de fluxos desses objetos, a matriz de distâncias dos locais, uma solução inicial e uma solução que serve como limite superior. O seguinte algoritmo descreve como foi feita seleção da solução de limite superior.

Algorithm 1 UpperBound

Input: N , F e D .

Output: Solução P .

```
Tome  $P$  uma solução aleatória
for 1 to  $N$  do
  Tome  $S$  uma solução aleatória
  while vizinhanca( $S$ ) for melhor do
     $S$  = vizinho melhor de  $S$ 
  end while
  if  $C(P) \leq C(S)$  then
     $P = S$ 
  end if
end for
```

O algoritmo do limite superior (UpperBound) toma uma solução aleatória P inicial e realiza uma busca local em N outras soluções aleatórias, observando se na vizinhança dessas soluções existe alguma que seja melhor. Essa busca na vizinhança é feita trocando dois objetos de posição (*Change2*), caso essa troca resulte em uma alocação melhor a vizinhança passa a ser feita nessa nova alocação, porém com menos objetos a serem trocados. Após encontrada uma solução boa, se ela for melhor que a solução inicial, a mesma é substituída. Essa comparação é feita N vezes. Como são N análises de soluções aleatórias e a busca na vizinhança são combinações 2 a 2, $N(N - 1)/2$, a complexidade do limite superior fica $O(n^3)$.

O algoritmo abaixo descreve o *Branch and Bound*, onde o mesmo faz um cálculo prévio do limite inferior antes de explorar um conjunto de alocações, para poder podar a árvore de alocações caso ultrapasse o limite superior. Nesse algoritmo a alocação

é feita inicialmente alocando apenas um objeto em um local e verificando o custo dessa alocação, caso esse custo somado com o limite inferior ultrapasse o limite superior, todas as possíveis alocações derivadas dessa alocação são ignoradas pelo algoritmo, caso contrário, o algoritmo explora até que não seja mais possível ou chegue em uma folha. Ao chegar na folha o algoritmo identifica se a alocação atual é melhor que o limite superior, se for atualiza esse limite para o custo da nova alocação. Ao final do procedimento recursivo, o algoritmo deve retornar a melhor solução de alocação e seu respectivo custo.

Algorithm 2 Branch and Bound

Input: N, F, D, S e P .

Output: Solução P e custo da solução $C(P)$.

```

if (Tempo de 3h) then
    return  $P$ 
end if
if ( $index == N$ ) then
    if  $C(S) \leq C(P)$  then
         $P = S$ 
    return  $P$ 
    end if
else
    for  $i = index$  to  $N$  do
         $S = troca(i, index, S)$ 
        novoF = Matriz de fluxo sem os objetos já alocados
        novoD = Matriz de distancias dos locais vazios
        custoMinimo = gilmoreLowler(novoF, novoD)
        if ( $custoMinimo + C(S) \leq C(P)$ ) then
            BranchAndBound( $N, novoF, novoD, S, P, index + 1$ )
        end if
         $destroca(i, index, S)$ 
    end for
end if

```

O algoritmo toma como parâmetros a quantidade N de objetos a serem alocados, as matrizes de fluxos F e de distâncias D , uma solução inicial S e a solução melhor até o momento P . O caso base ocorre quando o algoritmo tenta alocar o último objeto no último local. Se o custo de toda a alocação for melhor que o custo da alocação P , então P é atualizado para essa nova alocação. No caso recursivo, o algoritmo realiza um procedimento de troca, que significa alocar um objeto ainda não alocado no local $index$. Feito isso, o limite inferior é feito tomando como base as novas matrizes de fluxos e de distâncias, onde as mesmas só possuem, respectivamente, os fluxos entre os objetos que ainda não foram alocados e as distâncias entre os locais que ainda estão vazios. Esse limite é feito utilizando o Gilmore-Lowler Bound. Se o custo dos objetos que já foram alocados somado com o limite inferior não ultrapassar o limite superior, ocorre a recursão aumentando o $index$ em uma unidade, o que significa dizer que um objeto foi alocado e a próxima alocação deve ser feita no local seguinte. Depois de voltar da recursão, o algoritmo realiza a desalocação ($destroca(i, index, S)$) do objeto para poder alocar outro e repetir o processo para saber se é promissor. Além disso, foi definido um tempo limite

de 3h, para caso o algoritmo demore muito tempo para achar um solução.

A complexidade do cálculo do limite é $O(n^3)$, dada pela alocação linear. Entretanto, o cálculo do limite precisa que suas matrizes estejam com suas linhas previamente ordenadas, uma em ordem crescente e a outra em ordem decrescente, isso pode ser feito utilizando um algoritmo de ordenação como o QuickSort que ordena cada linha das duas matrizes, tendo assim, complexidade $O(n^2 \log n)$ para as ordenações.

4. Experimentos e Discussão dos Resultados

Para executar os experimentos foi utilizada a linguagem de programação C++ para implementação do algoritmo e o *Banchmark* QAPLib [2] com instâncias do Taillard com tamanhos de 12 a 20 objetos. Os testes foram feitos utilizando um processador Intel core I7 com 3.4 GHz e memória RAM de 8 GB. Fixou-se 3 horas (10800 segundos) para cada umas das instâncias testadas. Além disso, foi feito um comparativo com um algoritmo desenvolvido em Fortran por Burkard e Derigs [3], nesse não houve limite de tempo para execução das instâncias. A Tabela 1 mostra a comparação do tempo entre as duas implementações juntamente com o custo mínimo encontrado.

Tabela 1. Comparação entre exeperimentos

Instância	Tempo (Raul)	Tempo (Burkard e Derigs)	Custo Mínimo (Raul)	Custo Mínimo (Burkard e Derigs)
tai12a	619.65s	0.01s	224416	224416
tai12b	105.03s	2.82s	39464925	–
tai15a	–	10.22s	390782	388214
tai15b	–	3.38s	51765268	–
tai17a	–	123.87s	499878	491812
tai20a	–	22031.4s	719556	703482

Algumas instâncias não foram capazes de serem exibidas os custos ao serem testadas com o algoritmo de Burkard e Derigs, mas o programa foi capaz de exibir a alocação correta para a instância. Então, o resultado do tempo, para esse algoritmo, é válido.

O algoritmo desenvolvido não conseguiu se igualar ao algoritmo do Burkard e Derigs, pois apenas duas instâncias conseguiram executar em menos de 1 hora. Porém, as instâncias que ultrapassaram o tempo limite de 1 hora tiveram um resultado próximo ao algoritmo comparado. Além disso, foi possível observar que houve uma melhoria de tempo no algoritmo implementado entre as instâncias tai12a e tai12b, enquanto que para o algoritmo de Burkard e Derigs houve um aumento no tempo, isso pode ter sido causado devido o padrão de geração aleatório das instâncias do Taillard.

5. Conclusão

O algoritmo implementado, ao ser comparado com um algoritmo da literatura, se mostrou pouco eficiente, pois algumas instâncias não conseguiram executar em um tempo computacional menor que 3 horas e as instâncias que conseguiram, obtiveram tempo superior. Entretanto, o algoritmo é capaz de encontrar a alocação com menor custo para instâncias do PQA.

Como trabalhos futuros, uma implementação de um algoritmo heurístico será proposta, e a comparação poderá ser feita com essa implementação exata, com o objetivo de tentar aproximar mais as instâncias que não conseguiram executar por completo.

Referências

- [1] BENLIC, U. e HAO, J. (2013) **Breakout local search for the quadratic assignment problem**. Applied Mathematics and Computation 219, 4800-4815.
- [2] BURKARD, R.E., ÇELA, E., KARISCH, S.E. e RENDL, F. (2002) **QAPLIB - A Quadratic Assignment Problem Library**. Disponível em <<http://anjos.mgi.polymtl.ca/qaplib/>>. Acesso em 4 de Outubro de 2017.
- [3] BURKARD, R. E. e DERIGS, U. (1980) **Assignment and matching problems: Solution methods with fortran programs**. Lecture Notes in Economics and Mathematical Systems. Springer, Berlin. Vol. 184.
- [4] GILMORE, P. C. (1962) **Optimal and Suboptimal Algorithms for the Quadratic Assignment Problem**. Journal of the Society for Industrial and Applied Mathematics. Vol. 10, No. 2, pp. 305-313.
- [5] KOOPMANS, T. C. e BECKMANN, M. (1957) **Assignment Problems and the Location of Economic Activities**. Econometrica, Vol. 25, No. 1, pp. 53-76.
- [6] LAWLER, E. L. (1963) **The Quadratic Assignment Problem**. pp. 586-599.
- [7] NUGENT, C.E., VOLLMAN, T.E. e RUML, J. (1968) **An experimental comparison of techniques for the assignment of facilities to locations**. Operations Research, 16:150-173.
- [8] TAILLARD, E. (1990) **Robust taboo search for the quadratic assignment problem**. Ecole Polytechnique Fédérale de Lausanne, Département de Mathématiques, CH.1015 Lausanne. Switzerland.