Metaheurística Grasp aplicada na resolução de Problemas de Sequenciamento baseados em Grafo de Conflito

Raphael A. Silva[1], **Gustavo C. Menezes**[1]

[1] Centro Federal de Educação Tecnológica de Minas Gerais
(CEFET-MG) - Belo Horizonte - MG - Brazil

raphael.assis4347@gmail.com, gustavo@cefetmg.br

Abstract. *In several applications there is a need to schedule tasks on a production / transportation line and you want to obtain a schedule that minimizes the time spent to complete them. When these tasks conflict with each other, that is, when some of them cannot be performed at the same time as others, we have a complex problem. A specification of this problem occurs in transport systems such as port terminals that have a very complex cargo route system and that allow a large number of possible combinations. The purpose of this study is to seek GRASP-based Heuristic solutions to optimize these routes, thereby reducing the time to load and unload ships.*

Resumo. *Em diversas aplicações há a necessidade de agendar tarefas em uma linha de produção / transporte e deseja-se obter um agendamento que minimize o tempo gasto para concluí-las. Quando essas tarefas possuem conflitos entre si, isto é, quando algumas delas não podem ser executadas ao mesmo tempo que outras, temos um problema complexo. Uma especificação desse problema ocorre em sistemas de trasportes como terminais portuários que apresentam um sistema de rotas de cargas muito complexo e que possibilita uma grande quantidade de combinações possíveis. O objetivo desse estudo é buscar soluções Heurísticas baseadas em GRASP pra otimizar essas rotas, reduzindo assim o tempo para carregar e descarregar os navios.*

1. Introdução

O grande avanço no setor de transportes deu inicio a uma verdadeira revolução no processo de globalização. A demanda por capacidade e agilidade em entregas tornou os sistemas de transporte, incluindo o naval, extremamente complexo, demandando diariamente milhares de toneladas de produtos a serem transportados. Os terminais portuários são responsáveis por carregar e descarregar navios de carga e fornecer suporte de armazenamento para intermediar a troca da carga com outro meios de transporte como caminhões e trens. Devido a enorme quantidade de produtos carregados pelos navios a carga e descarga se torna extremamente demorada e torna bastante trabalhoso alocar os pátios de armazenamento para os produtos. Por esses motivos é de grande relevância que sejam desenvolvidos meios de otimizar as rotas dos produtos e o processo de alocação dos pátios, reduzindo o tempo para descarregar os navios. Assim, esses fatores motivam a busca de um algoritmo que gere um bom agendamento de rotas com conflitos.

A partir disso, busca-se desenvolver um algoritmo capaz de otimizar o agendamento das rotas com conflitos, reduzindo o tempo necessário para a conclusão das rotas.

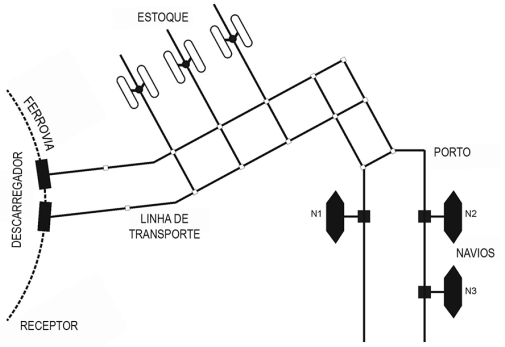
2. Detalhamento do problema do porto

Na figura 1a é representado o esquema de um terminal portuário. Os navios atracam nos cais e aguardam para serem carregados ou descarregados. Os navios podem ter seus produtos descarregados diretamente para um vagão de trem, para um caminhão ou para um pátio de armazenamento de carga. Um conjunto de equipamentos é instalado entre o cais, os pátios e os demais veículos terrestres para realizar o transporte dos produtos. A combinação entre os equipamentos utilizados para realizar esse transporte gera uma rota.

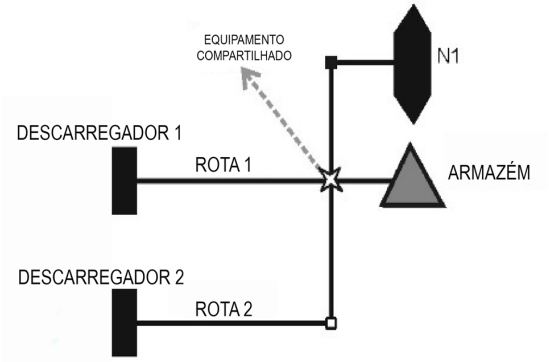
Os portos possuem uma grande quantidade de rotas para transportar os produtos e várias dessas rotas compartilham os mesmos equipamentos, impossibilitando a execução de ambas rotas simultaneamente, como é representado na figura 1b. Nesse caso, um dos produtos deve aguardar o outro ser transportado para em seguida poder continuar a sua rota.

Figura 1. Representação Porto

(a) Representação de um terminal portuário



(b) Representação das rotas em conflito



Além da complexidade em evitar os conflitos é desejável gerar um agendamento em que ocorra menos trocas de produtos nos pátios de armazenamento. Isto ocorre pois em portos que trabalham com produtos a granel, como soja, milho e demais grãos, é preciso lavar os pátios para armazenar outro produto no local. O mesmo ocorre com produtos do mesmo tipo, mas de purezas diferentes.

3. Revisão da literatura

O uso de heurísticas para otimizar o agendamemento de tarefas e serviços é extremamente recorrente na industria, no transporte e em instituições de prestação de serviço. Sellaro [Sellaro 2017] propõe o uso da técnica Particle Swarm Optimization para melhorar o agendamento de tarefas na integração de aplicações empresariais. Branco [Branco 2010] sugere o uso de um algoritmo genético em conjunto com a heurística job-shop para realizar o agendamento de tarefas em sistemas de manufatura. Em [Ozdamar and Yazgac 2010] é abordado o problema de gerenciamento de estoque utilizando uma abordagem heurística hierárquica.

O problema de agendamento de rotas com conflitos é abordado por Menezes [Menezes 2015] e Menezes, Mateus, Ravetti [Menezes et al. 2015] onde utiliza-se um algoritmo baseado em GRASP para agendar rotas de produtos nos terminais portuários. Essas pesquisas ilustram a grande aplicação de algoritmos de agendamento de tarefas.

4. Modelamento do problema

O modelo matemático proposto utiliza um grafo de conflito[1] para representar as incompatibilidades das rotas, onde os vértices representam as rotas dos produtos e as arestas representam que existe um conflito entre elas. Os conflitos serão medidos em termos de densidade e essa medida é feita calculando a razão entre o número de arestas no grafo pelo número de arestas de um grafo completo com o mesmo número de vértices. $densidade = \frac{2*arestas}{vertices \times (vertices-1)}$.

Uma vez estabelecido o modelo do problema analisamos o comportamento das soluções para diferentes quantidades de rotas e conflitos. Variando-se o número de rotas percebemos um aumento na complexidade do problema, pois assim mais possibilidades de agendamento serão possíveis. Analisando o problema quando se varia a densidade de conflitos notamos um pico na complexidade do problema para 50% de conflito. Quando a densidade de conflitos é muito pequena ou muito grande o agendamento é mais simples. No caso de não haver conflitos todas as rotas podem ser realizadas simultaneamente e o makespan[2] seria o tempo demandado pela rota mais demorada. Por outro lado, no caso em que todas as rotas tiverem conflitos entre si todas as possibilidades de agendamento possuirão o mesmo makespan, a soma dos tempos de execução de todas as rotas, pois só poderá haver uma única rota em funcionamento a cada momento, tornando trivial a solução. Esse comportamento de mantém para densidades não muito distantes dos extremos. Por esse motivo o maior desafio se encontra em realizar agendamentos em que a densidade de conflitos entre as rotas é intermediária.

Ao revisar a literatura [Menezes et al. 2015, Menezes 2015] vemos que esse é um problema NP-Completo[3], isto é, um problema que não pode ser tratado computacionalmente em tempo viável por algoritmos conhecidos. Assim, mudamos nossa estratégia para tentar encontrar uma solução boa em tempo viável ao invés da solução ótima.

5. Algoritmo Proposto

Para o problema abordado é proposto a utilização da meta-heurística GRASP (Greedy Randomized Adaptive Search Procedure)[da Silva et al. 2006]. O GRASP consiste em realizar perturbações na solução através de uma busca local de forma a diminuir o makespan. A busca local modifica a solução encontrada na construção utilizando um processo de melhoramento iterativo. Durante o processo de melhoramento da solução observa-se a variação do makespan encontrando e a partir do momento em que a solução encontrada deixa de melhorar, espera-se ter encontrado um mínimo local. Essa perturbação na solução é repetida até que um limite de iterações ou de tempo seja atingido.

Como o GRASP analisa apenas a variação do makespan não conseguimos encontrar garantidamente o mínimo global, pois não há como distinguir entre um mínimo global e local com esse método. Assim, o GRASP não nos garante uma solução ótima, apenas uma boa solução na vizinhança. Para contornar esse problema desenvolvemos uma busca global que gera uma pertubação na solução de forma alterar a vizinhança de

[1] Grafo de conflito é um grafo onde os vértices representam as rotas e as arestas representam conflitos entre elas.

[2] Makespan é o tempo total de agendamento, o tempo em que a ultima rota termina de ser executada.

[3] Problemas NP-Completos ou NP-Hard são problemas matemáticos que só possuem solução através de algoritmos com complexidade exponencial (não viável).

exploração. A repetição desse processo aumenta as chances de a solução convergir para o ótimo global.

O algoritmo de construção, algorithm 1, recebe uma lista que contém os índices das rotas, o tempo gasto para completar a rota, um booliano indicando se a rota foi agendada e os tempos de inicio e fim do agendamento, que inicialmente possuem um valor inválido. O primeiro passo consiste em realizar uma construção semi-gulosa em que utiliza-se o algoritmo de ordenação quicksort para ordenar as rotas de acordo com um método guloso. Foram desenvolvidos três métodos gulosos que selecionam as rotas de acordo com o tempo gasto, quantidade de conflitos e pelo produto tempo gasto por quantidade de conflitos. Em seguida, escolhe-se uma rota aleatoriamente para começar no tempo zero e realiza-se um agendamento guloso para as demais. Após a construção inicial, executa-se algumas iterações de refinamento da construção, alterando-se a rota inicial e reagendando as rotas seguindo o mesmo método guloso. Utiliza-se um parâmetro FB para limitar as iterações realizando-se N/FB iterações de refinamento, onde N é o número total de rotas a serem agendadas. O melhor agendamento desse processo é usado como a solução de construção.

Terminada a construção realiza-se uma busca local variando os tempos de inicio das rotas. Foram desenvolvidas três buscas locais e uma busca global para variar ao máximo as perturbações.

De forma geral, as buscas locais permutam as rotas no vetor de rotas reagendando a solução alterando o tempo de início das rotas de acordo com suas posições no vetor. Além do embaralhamento gerado para variar a solução, a relação de dependência entre as rotas gerada pelos conflitos acrescenta um fator de aleatoriedade às buscas locais. Desse modo, mesmo trocando somente a posição de duas rotas no vetor, o novo agendamento é significativamente diferente do anterior.

Na busca local a função EmbaralharRotas oculta a implementação da permutação realizada pela busca local. A seguir será abordado as permutações realizadas por cada busca. O parâmetro FatorGlobal controla o número de vezes que a busca global será solicitada no programa.

6. Busca Global

A busca global tem a finalidade de gerar um agendamento bastante diferente visando encontrar um outro mínimo local. Para essa finalidade, implementou-se o método de embaralhamento de Fisher–Yates proposta por Durstenfeld[Durstenfeld 1964] e Sattolo[Sattolo 1985, Wilson 2004]. A função de Durstenfeld gera uma permutação aleatória das rotas cujas sequencias possuem probabilidades iguais de ocorrerem. Sua complexidade é O(n) e apresenta variabilidade máxima nas permutações.

6.1. Buscas Locais

A busca local 1 resume-se em trocar as posições de duas rotas aleatoriamente. Sua complexidade é constante e sua variabilidade é alta. A busca local 2 é bastante semelhante com a busca local 1 e consiste em trocar duas rotas aleatoriamente, sendo uma delas a rota inicial. Sua complexidade é constante e sua variabilidade é alta. A busca local 3 realiza uma troca entre as rotas pares e ímpares entre si em cada iteração. A variabilidade é média e sua complexidade é constante.

Algorithm 1 ConstrucaoSimples

função CONSTRUCAOSIMPLES(N, rotas, agendamento)
    $makespan \leftarrow 0$
    para $i = 0$ to i **faça**N
        se $rotas(i).isAgendado = True$ **então**
            $continue$
        fim se
        $maxTempoConf \leftarrow 0$
        para $j = 0$ to n **faça**
            se $rotas(j).isAgendado = True$ **então**
                se $VerificaConflito(i, j) = True$ **então**
                    se $rotas(j).Tempofinal > maxTempoConf$ **então**
                        $maxTempConf \leftarrow rotas(j).Tempofinal$
                    fim se
                fim se
            fim se
        fim para
        $agendamento(i).index \leftarrow rotas(i).index$
        $rotas(i).Tempoinicio \leftarrow maxTempConf$
        $agendamento(i).Tempoinicio \leftarrow rotas(i).Tempoinicio$
        $rotas(i).Tempofinal \leftarrow maxTempConf + rotas(i).TempoGasto$
        $agendamento(i).Tempofinal \leftarrow rotas(i).Tempofinal$
        $rotas(i).isAgendado \leftarrow true$
        $agendamento(i).isAgendado \leftarrow true$
        se $rotas(i).Tempofinal > makespan$ **então**
            $makespan \leftarrow rotas(i).Tempofinal$
        fim se
    fim para
    $return\ makespan$
fim função

As buscas locais são repetidas até um limite passado pelo parâmetro (MaxIterations). Além disso, para aumentar a região de busca, a cada MaxIterations/FatorGlobal realiza-se uma iteração da busca global, colocando a solução em um novo mínimo local.

7. Experimentos computacionais

Para testar a eficiência do algoritmo foram geradas instancias com variados números de rotas e conflitos. Todos os testes computacionais foram realizados em um computador com processador Intel i7-7500U CPU @ 2.70GHz 2.90 GHz com 8 GB de RAM.

O primeiro teste realizado teve como objetivo selecionar o melhor método guloso para a construção. Para isso, comparou-se o makespan e o tempo de execução das construções que selecionavam as rotas pelo tempo, número de conflitos e pelo produto tempo por número de conflitos. O FB utilizado foi igual a 3, limitando o refinamento a testar 1/3 das rotas começando no tempo inicial.

Na tabela 1, Conflitos, tempo e tempo-conflito se referem aos métodos que anali-

Algorithm 2 Construcao

função CONSTRUCAO(N, rotas, FB)
$Quicksort(rotas, 0, N-1)$
$i \leftarrow numeroAleatorio()$
$rotas(i).isAgendado \leftarrow True$
$rotas(i).Tempoinicio \leftarrow 0$
$rotas(i).Tempofinal \leftarrow rotas(i).TempoGasto$
$Makespan \leftarrow ConstrucaoSimples(N, rotas, agendamento)$
$k \leftarrow 0$
enquanto $k < N/FBcont < MaxIteration$ **faça**
$makespan \leftarrow 0$
$ResetaAgendamento()$
$agendamento[k].Tempoinicio \leftarrow 0$
$agendamento[k].Tempofinal \leftarrow agendamento[k].TempoGasto$
$agendamento[k].isAgendado \leftarrow True$
$makespan \leftarrow ConstrucaoSimples(N, rotas, agendamento)$
se $makespan < Makespan$ **então**
$Makespan \leftarrow makespan$
$AtualizaAgendamento()$
fim se
$k \leftarrow k + 1$
fim enquanto
$return\ Makespan$
fim função

sam a quantidade de conflitos das rotas, tempo de execução da rota e o produto tempo por quantidade de conflitos, respectivamente. Notamos que o método guloso que analisa o tempo das rotas é mais vantajoso que os demais, visto que esse método gerou uma maior quantidade de makespans baixos e uma menor quantidade de makespans altos. Assim, esse método gerou o melhor makespan em 7 instancias e não apresentou um makespan pior que os demais métodos. Por esse motivo este método de construção foi selecionado para compor o algoritmo de sequenciamento. O tempo de execução dos métodos foram muito próximos e, como a complexidade algorítmica são iguais em ambos, não interferiu na escolha dos métodos.

Após a escolha da construção, realizou-se experimentos com as buscas locais com o objetivo de selecionar a melhor entre os três métodos propostos. Foi estabelecido um limite de 1000 iterações, um FatosGlobal igual a 5 e sem refinamento da solução de construção.

Analisando os resultados, observamos que a busca local 1 é mais adequada devido as soluções encontradas. Essa busca gerou o makespan mais baixo entre os três métodos em sete instancias e o pior makespan em apenas duas delas, enquanto as demais apresentaram menos soluções melhores e mais soluções piores. O tempo de execução dos métodos foram muito próximos e, como a complexidade algorítmica são de mesma ordem em ambos, não interferiu na escolha dos métodos.

Por fim, realizamos um experimento computacional para comparar as soluções

Algorithm 3 Busca Local

função BUSCA LOCAL(N, rotas, agendamento, MaxIterations, FatorGlobal)
  enquanto $cont < MaxIteration$ **faça**
    se $cont \,\%\, (MaxIterations/FatorGlobal) = 0$ **então**
      $BuscaGlobal(rotas, N, agendamento)$
    senão
      $EmbaralharRotas(rotas, N, agendamento)$
    fim se
    $ResetaAgendamento()$
    $makespan \leftarrow ConstrucaoSimples(N, rotas, agendamento)$
    se $makespan < Makespan$ **então**
      $Makespan \leftarrow makespan$
      $AtualizaAgendamento()$
    fim se
    $cont \leftarrow cont + 1$
  fim enquanto
fim função

encontradas pelo algoritmo com a solução ótima. Utilizou-se um limite de 10 iterações, um FB igual a 2 e FatorGlobal igual a 2.

Tabela 3. GRASP x Ótimo

Rotas	Conflitos (%)	GRASP	Ótimo
10	25	27	27
10	50	38	35
10	75	41	40

Este último experimento foi condicionado de forma a limitar bastante as buscas a fim de observar o makespan gerado por buscas com pouca profundidade. Em problemas grandes é inviável realizar buscas profundas e por isso esse tipo de teste é interessante. Contudo, podemos observar que o resultado obtido é bastante significativo. Apesar da instancia ser extremamente pequena, para possibilitar calcular a solução ótima, um bom makespan foi alcançado com apenas 8 iterações.

8. Conclusão

Os experimentos computacionais mostram que o algoritmo proposto é eficiente e rápido para instancias pequenas e uma boa opção para instancias grandes. Sua complexidade é O(n^3) e é bastante simples de ser utilizado. Desse modo, qualquer problema que envolva sequenciar e agendar tarefas, sejam elas com ou sem conflitos, esse algoritmo apresentará bons resultados, bastando ajustar adequadamente os parâmetros FB, FatorGlobal e MaxIteration para atingir uma boa variabilidade de busca.

As aplicações para esse algoritmo são bastante amplas e comuns. Nos terminais portuários espera-se utilizá-lo para otimizar as rotas dos produtos para agilizar o processo

Tabela 1. Métodos gulosos

Rotas	Conflitos (%)	Conflitos	Tempo	Tempo-Conflito
100	25	264	256	207
100	50	375	400	403
100	75	519	501	484
500	25	1415	1320	1325
500	50	2047	1926	2110
500	75	2641	2404	2411
1000	25	2822	2570	2616
1000	50	3979	3803	3812
1000	75	5567	5279	5253
5000	25	13875	13032	12961
5000	50	21309	20273	20283
5000	75	27168	25923	25931

Tabela 2. Buscas Locais

Rotas	Conflitos (%)	Busca 1	Busca 2	Busca 3
100	25	210	216	236
100	50	337	329	340
100	75	502	480	509
500	25	1348	1271	1362
500	50	1985	2020	2016
500	75	2504	2563	2629
1000	25	2543	2697	2707
1000	50	4011	4019	4087
1000	75	5373	5410	5388
5000	25	13791	13634	13601
5000	50	21215	21265	21130
5000	75	27188	27172	27109

de carregamento e descarregamento dos navios. De forma semelhante, o algoritmo pode ser utilizado para agendar cirurgias em hospitais, apresentando conflitos entre os recursos hospitalares tais como bolsas de sangue, remédios, disponibilidade de salas, médicos e etc; Agendamento de encomendas para transportadoras apresentando conflitos pela disponibilidade de caminhões e espaço de carga; Agendamento de turmas escolares e atividades relacionas; e qualquer problema de sequenciamento de tarefas/serviços.

Referências

Branco, R. M. (2010). Agendamento de tarefas em sistemas de manufatura job-shop realista com demanda por encomenda: Solução por algoritmo genético. Master's thesis, Universidade Federal de Santa Catarina.

da Silva, G. C., Ochi, L. S., and Martins, S. L. (2006). Proposta e avaliação de heurísticas grasp para o problema da diversidade máxima.

Durstenfeld, R. (1964). Algorithm 235: Random permutation.

Menezes, G. C. (2015). Modelo e algoritmos para um problema integrado de planejamento, sequenciamento e alocação de pátios.

Menezes, G. C., Mateus, G. R., and Ravetti, M. G. (2015). Scheduling with incompatible jobs: model and algorithms.

Ozdamar, L. and Yazgac, T. (2010). A hierarchical planning approach for a production-distribution system. pages 37–16.

Sattolo, S. (1985). An algorithm to generate a random cyclic permutation.

Sellaro, D. F. (2017). Particle swarm optimization para agendamento de tarefas na integração de aplicações empresariais. pages 8–10.

Wilson, M. (2004). Overview of sattolo's algorithm.