Bruno Coswig Fiss
Kauê Soares da Silveira

GRASP aplicado ao problema de aterrissagem de aviões

INF05010 – Otimização Combinatória

Professor: Marcus Rolf Peter Ritt

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL

1º de julho de 2010

Sumário

1 Descrição do problema p. 3

1.1 Descrição formal . . . . p. 3

1.2 Formulação como um programa inteiro . . . . p. 3

1.3 Representação de um solução . . . . p. 4

2 Algoritmo proposto p. 5

2.1 Idéia geral . . . . p. 5

2.2 GRASP . . . . p. 5

2.2.1 Criação de soluções . . . . p. 5

2.2.2 Vizinhança e busca local . . . . p. 6

3 Experimentos p. 7

3.1 Configurações . . . . p. 7

3.2 Resultados . . . . p. 7

3.3 Análise . . . . p. 7

Referências Bibliográficas p. 8

1 Descrição do problema

1.1 Descrição formal

O problema de aterrissagem de aviões consiste em definir um momento no tempo para a aterrissagem de cada avião $i \in P$, sendo P o conjunto de aviões. Cada avião possui os seguintes dados:

E_i: Momento mais prematuro em que o avião i pode realizar pouso.

T_i: Momento ideal para pouso do avião i.

L_i: Momento mais tardio em que o avião i pode realizar pouso.

g_i: Penalidade por unidade de tempo da diferença do pouso para o tempo ideal se o avião chegar mais cedo do que o ideal.

h_i: Penalidade por unidade de tempo da diferença do pouso para o tempo ideal se o avião chegar mais tarde do que o ideal.

S_{ij}: Distância de tempo requerida após o pouso do avião i para que o avião j possa pousar.

O objetivo é encontrar uma solução com somatório de todas as penalidades mais baixo possivel.

1.2 Formulação como um programa inteiro

O seguinte programa inteiro descreve o problema acima descrito:

$$
\begin{array}{rll}
\textbf{minimiza} & \sum_{i \in P} g_i \alpha_i + h_i \beta_i & \\
\textbf{sujeito a} & x_i = -\alpha_i + \beta_i + T_i, & \forall i \in P \\
& E_i \leq x_i \leq L_i, & \forall i \in P \\
& x_j - x_i \geq S_{ij}\delta_{ij} + (E_j - L_i)\delta_{ji}, & \forall i, j \in P \\
& \delta_{ij} + \delta_{ji} = 1, & \forall i, j \in P \\
& x_i \geq 0, x_i \in \mathbb{R}, & \forall i \in P \\
& \alpha_i \geq 0, \alpha_i \in \mathbb{R}, & \forall i \in P \\
& \beta_i \geq 0, \beta_i \in \mathbb{R}, & \forall i \in P \\
& \delta_{ij} \in \mathbb{B}, & \forall i, j \in P
\end{array}
$$

A variável x_i representa o momento da aterrissagem do avião i. As variáveis α_i e β_i indicam a diferença para menos ou mais, respectivamente, do momento da aterrissagem para o momento ideal de aterrissagem do avião i. δ_{ij} é uma variável binária que indica se o avião i aterrisa antes do avião j.

A formulação do programa inteiro foi realizada pelos autores antes da leitura de artigos relacionados com o problema, que por sua vez continham uma formulção similar. Para facilitar a compreensão, tornamos a simbologia utilizada por nós semelhante à utilizada nesses artigos [1, 2].

1.3 Representação de um solução

O programa inteiro que descreve o problema aqui tratado é misto, possuindo variáveis reais além de inteiras. Como o objetivo desse trabalho é aplicar uma meta-heurística a um problema inteiro, separamos a solução em duas partes, uma inteira e outra real.

A parte inteira da solução é a que contém as variáveis δ_{ij}, que descrevem, em conjunto, a ordem de chegada dos aviões. O caminho inverso também é válido, ou seja, uma determinada ordem de chegada dos aviões descreve completamente as variáveis δ_{ij}. Já a parte linear contém as demais variáveis.

Portanto, representaremos uma solução como uma sequência de aviões que descreve a ordem de chegada desses. Após obter a solução inteira do problema, basta resolver o programa linear restante para obter a solução completa do problema.

2 Algoritmo proposto

2.1 Idéia geral

Nosso algoritmo utiliza a biblioteca GLPK (GNU linear programming kit) para resolução das partes lineares do problema sendo tratado.

Primeiramente, definimos o problema inteiro utilizando os dados lidos da entrada e as funções para criação de restrições disponibilizadas pela biblioteca GLPK. Após esse passo é possível utilizar o resolvedor interno do GLPK para resolver o problema inteiro, utilizando o método branch-and-cut. Essa característica foi requisitada na especificação do trabalho da disciplina, e também permite a comparação entre as soluções atingidas pelo algoritmo do GLPK e pelo método GRASP, além da comparação entre o desempenho na obtenção dessas.

Após isso, utilizamos a meta-heurística GRASP para criar soluções para a parte inteira do problema, ou seja, sequências de aviões que definem a ordem de chegada desses. Para cada solução inteira, modificamos as restrições do problema inteiro criado no primeiro passo de forma que ele respeite a ordem presente na sequência e torne-se, por consequência, linear. Cada programa linear gerado é então resolvido através do método simplex utilizando uma rotina da biblioteca GLPK.

Isso é feito para cada solução inteira gerada pelo método GRASP, tanto para as soluções geradas durante a construção gulosa aleatória quanto para as soluções geradas na busca local (soluções vizinhas da solução atual). A melhor solução encontrada é armazenada e impressa ao fim do algoritmo.

2.2 GRASP

2.2.1 Criação de soluções

A utilização de uma meta-heurística como o GRASP tem como primeira etapa a definição dos conjuntos, funções e parâmetros necessários para a definição do problema combinatorial a ser resolvido. Utilizaremos os conjuntos, funções e parâmetros citados no artigo de Resende e Ribeiro [3] para formalização do nosso método.

Seguindo o artigo, os conjuntos $E, F \subseteq 2^E$ e a função $f : 2^E \to \mathbb{R}$ foram definidos. O conjunto E de elementos que possivelmente fazem parte da solução é o conjunto de variáveis δ_{ij}. O conjunto F é o conjunto de todos os conjuntos de variáveis δ_{ij} tal que, caso as variáveis do conjunto sejam todas definidas como tendo valor 1, e as variáveis δ_{ij} que não estiverem no conjunto forem definidas como tendo valor 0, existe uma solução para o programa linear resultante. A função de custo f mapeia um dado conjunto de variáveis δ_{ij} para o custo da solução mínima do programa linear em que as variáveis pertencentes ao conjunto são definidas como 1 e as demais como 0.

Além disso, as meta-construções do algoritmo GRASP precisam ser instanciadas, e essas são a representação da solução e de cada elemento a ser adicionado a essa, a RCL (Restricted Candidate List), e o parâmetro α, além dos parâmetros para controle do término do algoritmo, que aqui são $iter_{max}$ e $time_{max}$.

A representação de uma solução será a ordem da chegada dos aviões, que por sua vez será representada por uma sequência de aviões, na qual um avião chega antes de outro se estiver mais à esquerda na sequência. Os elementos a serem adicionados, passo a passo, para a construção da solução, são os proprios aviões. Dado um parâmetro α, em um determinado passo da construção da solução, nossa RCL é o conjunto de aviões que estão posicionados, na sequência ideal, a até α posições da posição a ser adicionada na solução atual e que, se forem adicionados, não tornarão a solução inviável. E.g., dada a sequência ideal 1,2,6,4,5,3, o parâmetro $\alpha = 2$ e a solução atual ser 1,4, a RCL seria $\{6,2,5\}$ desde que a adição de qualquer um desses elementos à solução atual mantivesse-a viável.

A sequência ideal é definida inicialmente como a ordem de chegada dos aviões caso todos eles chegassem no seu tempo ideal. Se uma dada solução obtiver um custo menor do que a sequência ideal, a sequência que define a nova solução se tornará a sequência ideal.

Para efetivamente testar a sequência ideal, nossa primeira iteração utiliza $\alpha = 0$, ou seja, uma construção totalmente gulosa.

Após a realização de testes e do estudo da variação Reactive GRASP da meta-heurística GRASP, decidimos implementar parte das variações no nosso algoritmo. A variação implementada é simples: o parâmetro α é, a cada iteração, sorteado aleatoriamente entre 0 e um novo parâmetro, chamado α_{max}.

Os parâmetros $iter_{max}$ e $time_{max}$ definem, respectivamente, o número máximo de iterações da e o tempo máximo em segundos gastos na heurística GRASP.

2.2.2 Vizinhança e busca local

A busca local pode ser feita de duas formas: com o primeiro incremento ou com o melhor incremento. Decidimos utilizar o primeiro incremento, ou seja, pesquisamos a vizinhança de uma dada solução em busca de uma solução melhor e, quando encontramos a primeira, passamos a considerar essa nova solução encontrada como a solução atual, buscando agora uma solução melhor na vizinhança da atual.

Resta definirmos o que é a vizinhança de uma solução, o que é feito a seguir: uma dada solução, representada por uma sequência de aviões na qual dois aviões adjacentes foram trocados, um sendo colocado no lugar do outro, e nenhuma outra modificação ocorrendo, é vizinha da solução original, sem a troca dos aviões. A vizinhança de uma dada solução é simplesmente o conjunto de todos os vizinhos dessa.

3 *Experimentos*

3.1 Configurações

Os testes realizados tiveram por objetivo testar a eficácia da nossa abordagem dadas diferentes sementes aleatórias e parâmetros α_{max}. Testamos, é claro, esses parâmetros para cada caso de entrada.

Para repetir os testes, basta compilar o arquivo avioes.c, ligá-lo à biblioteca GLPK criando um executável chamado Avioes.exe, compilar o arquivo experimentos.c, e executar o arquivo resultante da ligação e montagem do objeto proveninte da compilação de experimentos.c. As compilações podem ser realizadas com o compilador gcc. A ligação e montagem dependem do sistema operacional utilizado, além da localização da biblioteca GLPK.

3.2 Resultados

Em todos os testes o limite de iterações do método foi 100 e o limite de tempo foi de 20 segundos. Vários casos passaram de 20 segundos pois o limite de tempo so é verificado ao final de cada iteração, que pode ser longa.

3.3 Análise

Como pode-se perceber, a abordagem gulosa é extremamente eficaz na resolução desse problema, sendo que a utilização de $\alpha = 0$ na geração de uma solução gera a melhor solução ou um vizinho próximo dessa em 7 dos 8 casos de entrada testados. No caso airland2, a solução inicial gera uma solução com custo de 1500, porém com algumas iterações do método, utilizando um α variável, a melhor solução, com custo de 1480, é encontrada.

Pode-se concluir que o método é bastante afetado pela escolha da função que determina o custo incremental da adição de um dado elemento à solução que está sendo gerada, ou seja, a parte gulosa do algoritmo. No caso desse trabalho, a função, que determina esse custo pela posição do avião na sequência ideal, parece ter sido bem escolhida, o que gerou soluções ótimas em todos os casos.

Além disso, o parâmetro α é vital, o que nos levou a decidir não utilizar o método GRASP padrão, mas uma variação do Reactive GRASP, que decide o valor de α_{max} de forma aleatória com probabilidade uniforme para todos os α entre 0 e $\alpha_{max} - 1$. Essa escolha também parece ter ajudado no sucesso da abordagem.

Referências Bibliográficas

[1] J. E. Beasley, M. Krishnamoorthy, Y. M. Sharaiha, and D. Abramson. Scheduling aircraft landings–the static case. *Transportation Science*, 34(2):180–197, 2000.

[2] J. E. Beasley, M. Krishnamoorthy, Y. M. Sharaiha, and D. Abramson. Displacement problem and dynamically scheduling aircraft landings. *Transportation Science*, 55(1):54–64, 2004.

[3] Mauricio G. C. Resende, Mauricio G. C, Resende, and Celso C. Ribeiro. Greedy randomized adaptive search procedures, 2002.