UMA HEURÍSTICA GRASP PARA OTIMIZAR MAPAS DE SÍMBOLOS PROPORCIONAIS

Rafael Ghussn Cano, Cid Carvalho de Souza e Pedro Jussieu de Rezende
Universidade Estadual de Campinas — Instituto de Computação
Avenida Albert Einstein, 1251, CEP 13083-852 Campinas, SP, Brasil
rgcano@gmail.com, {cid, rezende}@ic.unicamp.br

RESUMO

Mapas de símbolos proporcionais são ferramentas cartográficas utilizadas para visualizar eventos associados a uma localização e a uma intensidade. Cada evento é representado por um símbolo, cuja área é proporcional à intensidade registrada. Os símbolos utilizados aqui são discos opacos e encontram-se organizados na forma de uma pilha. Quando dois ou mais discos se sobrepõem, partes deles podem ficar encobertas e pode ser difícil determinar seus tamanhos. Logo, a ordem na qual os discos são desenhados afeta a qualidade do mapa. Nesse projeto de iniciação científica, desenvolvemos uma heurística sofisticada baseada em GRASP com *path-relinking* para o problema de encontrar um empilhamento que maximize a soma dos comprimentos das bordas visíveis dos discos. Foram realizados experimentos com instâncias reais e a comparação dos resultados obtidos com as soluções ótimas confirma a qualidade de nossa heurística.

PALAVRAS CHAVE. GRASP, Geometria Computacional, Mapas de Símbolos Proporcionais.

Área principal: Metaheurísticas

ABSTRACT

Proportional symbol maps are a cartographic tool that employs symbols to represent data associated with specific locations. Each symbol is drawn at a location and its size is proportional to the numerical data collected at that point on the map. The symbols used here are opaque disks, which are stacked on top of each other. A portion of a disk might not be visible when it overlaps other disks. When large portions of a disk are covered, it is difficult to gauge its size. Therefore, the order in which the disks are drawn affects the visual quality of a map. In this work, we present a sophisticated heuristic based on GRASP with path-relinking for the problem of finding a stacking order that maximizes the total visible boundary of all disks. We experimented with real-world instances and the comparison against optimal solutions confirms the high quality of our heuristic.

KEYWORDS. GRASP, Computational Geometry, Proportional Symbol Maps.

Main area: Metaheuristics

1 Introdução

Mapas de símbolos proporcionais são ferramentas cartográficas utilizadas para visualizar dados ou eventos, tais como a magnitude de terremotos e a população de cidades. A cada evento associa-se uma localização e uma intensidade e a representação no mapa se dá através de um símbolo, cuja área é proporcional à intensidade registrada. Geralmente, os símbolos utilizados são discos, quadrados ou triângulos, que podem ser transparentes ou opacos. Neste trabalho, os símbolos considerados são discos opacos com centros no local do evento que cada um representa. Quando dois ou mais discos se sobrepõem, partes deles podem ficar encobertas e pode ser difícil determinar seus tamanhos. Logo, encontrar a ordem na qual eles devem ser dispostos é fundamental para que as informações sejam interpretadas corretamente. A Figura 1 mostra dois mapas representando a população das maiores cidades do nordeste do Brasil. O mapa à esquerda torna difícil a análise dos dados, pois alguns discos foram totalmente ou quase totalmente cobertos. Já a disposição dos discos no mapa à direita transmite as informações de forma muito mais eficiente.

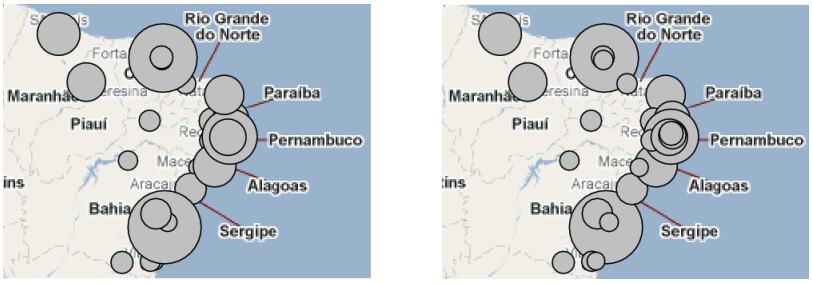

Figura 1: Mapas representando a população de algumas cidades da região nordeste do Brasil.

Seja S um conjunto de n discos e $\mathscr{A}$ o arranjo formado pelas bordas dos discos em S. Uma intersecção das bordas de dois ou mais discos define um *vértice* de $\mathscr{A}$. Um *arco* é a porção da borda de um disco que conecta dois vértices e não contém nenhum outro vértice. Um *desenho* de S é um subconjunto dos arcos e vértices de $\mathscr{A}$ desenhado sobre os interiores preenchidos dos discos de S. Neste projeto, consideramos os desenhos que correspondem a *empilhamentos*, isto é, os discos são organizados na forma de uma pilha e desenhados em seqüência, começando por aquele no último nível. Um bom desenho deve permitir um julgamento preciso dos tamanhos e da localização dos discos. Como ilustrado na Figura 2, esse julgamento depende do perímetro e não da área visível de cada disco. Assim, dois valores podem ser utilizados para medir a qualidade de um desenho: a menor borda visível de todos os discos (i.e., $\min_{1\leq i\leq n}\{\text{comprimento da borda visível do disco } i\}$) e a soma dos comprimentos das bordas visíveis de todos os discos. Os problemas *Max-Min* e *Max-Total* consistem em maximizar o primeiro e o segundo valor, respectivamente.

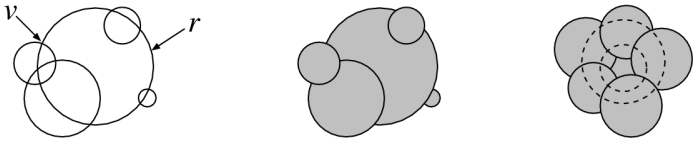

Figura 2: Um arranjo $\mathscr{A}$ com um vértice v e um arco r (esq.), um desenho dos discos em $\mathscr{A}$ (centro) e um desenho no qual o perímetro é mais importante que a área visível (dir.).

Cabello et al. (2010) apresentaram um algoritmo guloso de tempo $O(n^2 \log n)$ para resolver o problema Max-Min. Eles também mostraram que se todo ponto no plano está contido em $O(1)$ discos, a solução pode ser encontrada em tempo $O(n \log n)$. Kunigami et al. (2010) propuseram um modelo de programação linear inteira para o problema Max-Total e introduziram técnicas de decomposição para dividir os discos de uma instância em componentes menores, as quais podem ser resolvidas independentemente. Eles também forneceram as primeiras soluções comprovadamente ótimas para algumas instâncias reais.

A complexidade computacional do problema Max-Total para empilhamentos ainda está

em aberto e, portanto, é interessante o desenvolvimento de heurísticas que encontrem soluções de alta qualidade rapidamente. Em particular, a metaheurística GRASP já foi aplicada com sucesso a diversos problemas de otimização combinatória. Uma descrição das principais técnicas existentes na literatura pode se encontrada no trabalho de Resende e Ribeiro (2009). Cada iteração do GRASP é formada por duas fases: construção, que cria uma solução inicial, e busca local, que examina a vizinhança da solução inicial tentando obter melhorias. O procedimento básico do GRASP pode ser melhorado com a aplicação de *path-relinking*, uma estratégia de intensificação proposta por Glover (1996) que explora trajetórias entre soluções de alta qualidade. O uso de *path-relinking* com GRASP foi proposto pela primeira vez por Laguna e Martí (1999).

Neste projeto de iniciação científica, foi desenvolvida uma heurística baseada em GRASP com *path-relinking* para o problema Max-Total para empilhamentos, a qual inclui diversas técnicas avançadas descritas na literatura para esse procedimento. A comparação dos resultados obtidos pela heurística com as soluções ótimas confirma a qualidade dos métodos implementados. Não é de nosso conhecimento a existência de outras metaheurísticas propostas na literatura para esse problema. Nas próximas seções, descremos as principais estratégias utilizadas na heurística desenvolvida. A Seção 2 apresenta o algoritmo empregado na fase de construção do GRASP e a Seção 3 descreve as vizinhanças para a busca local. A implementação do *path-relinking* é apresentada na Seção 4. Na Seção 5, fornecemos alguns resultados computacionais.

2 Construção da solução inicial

As soluções iniciais de cada iteração do GRASP foram criadas com uma variação do algoritmo guloso proposto por Cabello et al. (2010) para o problema Max-Min. Os autores comparam esse algoritmo com outras três heurísticas simples para o problema Max-Total e os resultados experimentais reportados mostram que ele obtém soluções de maior qualidade no conjunto de instâncias considerado. Por essa razão, ele foi escolhido para a fase de construção. O algoritmo usa uma variação de uma árvore de segmentos, a qual descrevemos brevemente aqui (veja de Berg et al. (2008) para mais detalhes sobre árvores de segmentos).

Uma árvore T_i é construída para cada disco $i \in S$. Cada nó v de T_i armazena um intervalo $int(v)$ e dois valores $counter(v)$ e $vis\text{-}int(v)$. Se v é uma folha, $int(v)$ corresponde ao intervalo definido por um dos arcos na borda de i, caso contrário, $int(v)$ é a união dos intervalos $int(..)$ dos filhos de v. Em cada nó interno ou folha, $counter(v)$ armazena o número de discos que contém $int(v)$ mas não contém $int(pai(v))$. Em $vis\text{-}int(v)$ é guardado o comprimento da porção de $int(v)$ que estaria visível caso apenas intervalos de discos que ocorrem na subárvore com raiz em v cobrissem partes de $int(v)$. Na raiz, $vis\text{-}int(raiz(T_i))$ corresponde ao comprimento da borda visível do disco i.

Denotamos por I_i^j o intervalo na borda do disco i que está contido no interior do disco j. Quando há uma intersecção entre dois discos i e j e j está acima de i na solução corrente, I_i^j é inserido em T_i. Se i é movido para uma posição acima de j, deve-se inserir I_j^i em T_j e remover I_i^j de T_i. A inserção ou remoção de I_i^j atualiza todos os nós v de T_i tais que $int(v) \subseteq I_i^j$ mas $int(pai(v)) \not\subseteq I_i^j$. Os nós que satisfazem essa condição podem ser encontrados antes do início das iterações do GRASP, acelerando a execução da heurística. Para inserir o intervalo, incrementa-se $counter(v)$ e, caso este fosse igual a zero, atribui-se zero a $vis\text{-}int(v)$. Para remover o intervalo, decrementa-se $counter(v)$ e, caso seu novo valor seja zero, atribui-se a $vis\text{-}int(v)$ a soma dos valores $vis\text{-}int(..)$ dos filhos de v. Pode-se mostrar que inserções e remoções levam tempo $O(\log n)$.

O algoritmo para resolver o problema Max-Min começa com uma pilha de discos vazia. Dado um disco i que ainda não foi empilhado, seja b_i o comprimento da borda visível de i caso ele seja inserido abaixo dos discos que ainda não foram empilhados e acima dos demais. Cada iteração calcula os valores b_i para todos os discos i que ainda não estão na pilha e o disco com maior valor é empilhado. Esse procedimento é repetido até que todos os discos estejam na pilha. As árvores de segmentos são utilizadas da seguinte forma. Inicialmente, insere-se em cada árvore T_i os intervalos de todos os discos que interceptam i, como se i estivesse no nível mais baixo da pilha. Assim, b_i

é obtido examinando-se $vis\text{-}int(raiz(T_i))$. O disco com o maior desses valores é empilhado e para cada disco j que o intercepta, remove-se de T_j o intervalo na borda de j que está contido no interior do disco escolhido. Os demais discos são empilhados recursivamente.

Uma componente aleatória foi adicionada ao algoritmo descrito para adaptá-lo ao GRASP. A cada iteração, ao invés de empilharmos o disco i com maior valor de b_i, criamos uma *lista restrita de candidatos* (LRC) a partir de um parâmetro $\alpha \in [0,1]$. Sejam b^{min} e b^{max} o menor e o maior valor de b_i para todos os discos i que ainda não estão na pilha, respectivamente. Um disco i é inserido na LRC se $b_i \geq b^{min} + \alpha(b^{max} - b^{min})$. Um valor adequado para α deve criar soluções de alta qualidade e também gerar variabilidade nas soluções iniciais. Foram realizados experimentos com estratégias que utilizam valores fixos e variáveis de α. Após a criação da LRC, um dos discos pertencentes a ela é escolhido aleatoriamente usando-se funções de viés (*bias functions*). Uma descrição dessas funções, bem como das estratégias existentes na literatura para a seleção do valor de α se encontra no trabalho de Resende e Ribeiro (2009).

3 Busca local

Nesta seção, descrevemos duas vizinhanças de tamanho $O(n^2)$ denotadas por *vizinhança de inserção* e *vizinhança de troca*. Elas foram usadas para implementar o método *Variable Neighborhood Descent* (VND), o qual inicialmente realiza uma busca com a vizinhança de inserção, passando posteriormente à de troca quando nenhuma melhoria é obtida. No restante do texto, dada uma solução x, denotamos por $x(p)$ o disco no nível p de x e por $x^{-1}(i)$ o nível do disco i em x.

3.1 Vizinhança de inserção

Na vizinhança de inserção, um movimento consiste em remover um disco do empilhamento e inseri-lo em outra posição. Dado um disco i e um nível p, denotamos por $I(i,p)$ a inserção de i no nível p. Para a busca com essa vizinhança, um disco $i \in S$ é selecionado e todas as inserções de i são avaliadas. Se a melhor inserção encontrada produz uma melhoria, ela é executada. Caso contrário, o procedimento é repetido com outro disco, até que nenhuma inserção produza melhorias.

As inserções de um disco i são avaliadas com as árvores de segmentos. Suponha que i e j são dois discos consecutivos no empilhamento relativo a uma solução x, com $x^{-1}(j) > x^{-1}(i)$. Para obter a variação no valor da função objetivo causada pela troca de posição de i e j, removemos I_i^j de T_i e inserimos I_j^i em T_j. A variação é obtida examinando-se $vis\text{-}int(raiz(T_i))$ e $vis\text{-}int(raiz(T_j))$ antes e depois das alterações em T_i e T_j (essa variação é igual a zero quando i e j não se interceptam). Dessa forma, as inserções $I(i,x^{-1}(i)+1), I(i,x^{-1}(i)+2), \ldots, I(i,n)$ podem ser avaliadas rapidamente, em seqüência. Em seguida, restaura-se o estado original das árvores que foram modificadas e a mesma estratégia é usada para avaliar $I(i,x^{-1}(i)-1), I(i,x^{-1}(i)-2), \ldots, I(i,1)$.

3.2 Vizinhança de troca

Na vizinhança de troca, um movimento consiste em trocar a posição de dois discos em uma solução. Dados dois discos i e j, denotamos por $T(i,j)$ a troca de posição de i e j. Embora as árvores de segmentos possam ser usadas para avaliar movimentos nessa vizinhança, experimentos preliminares mostraram que essa é uma estratégia custosa e, por isso, outro método foi desenvolvido. A seguinte notação será utilizada: dado um arco r, seu comprimento é denotado por l_r e o disco em cuja borda ele se encontra é denotado por d_r.

Dada uma solução x, a troca $T(i,j)$ pode ser vista como um par de inserções consecutivas, uma de i no nível $x^{-1}(j)$ e outra de j no nível $x^{-1}(i)$. Para avaliar $T(i,j)$, consideramos que as duas inserções são independentes (isto é, ignoramos os efeitos da primeira sobre a segunda) e fazemos as correções necessárias para que o valor desejado seja obtido. Duas matrizes A e B de dimensão $n \times n$ são criadas, as quais armazenam o valor de todas as inserções e as correções, respectivamente. Cada elemento $A[i,j]$ armazena a variação no valor da solução causada pela inserção $I(i,x^{-1}(j))$. A matriz B é inicializada em zero e, para cada arco r, l_r é somado ou subtraído de alguns elementos de B, de forma que $A[i,j]+A[j,i]+B[i,j]+B[j,i]$ forneça a variação correta causada por $T(i,j)$.

Considere a troca $T(i,j)$ e suponha que $x^{-1}(j) > x^{-1}(i)$. Para construir a matriz B, três casos devem ser observados:

1. Seja r um arco que não está na borda de i nem de j, tal que $x^{-1}(j) > x^{-1}(d_r) > x^{-1}(i)$. Se i e j contém r e não há nenhum outro disco acima de d_r que contém r, a inserção $I(j,x^{-1}(i))$ descobre r. Como r deve permanecer coberto após $T(i,j)$, l_r deve ser subtraído de $B[i,j]$.
2. Seja r um arco visível na borda de j. Se i contém r, as inserções $I(i,x^{-1}(j))$ e $I(j,x^{-1}(i))$ cobrem r e a soma $A[i,j]+A[j,i]$ considera l_r duas vezes. Assim, l_r deve ser somado a $B[i,j]$.
3. Seja r um arco na borda de i e suponha que j é o único disco acima de i que contém r. As inserções $I(i,x^{-1}(j))$ e $I(j,x^{-1}(i))$ descobrem r e a soma $A[i,j]+A[j,i]$ considera l_r duas vezes. Assim, l_r deve ser subtraído de $B[i,j]$.

A matriz A pode ser construída avaliando-se todas as inserções dos discos em S. A matriz B é obtida identificando-se os três casos já descritos e realizando-se as operações necessárias em cada caso. Sempre que um movimento é executado, as matrizes devem ser atualizadas. Se i e j são os discos trocados no último movimento, deve-se examinar os arcos na borda de i e de j, os arcos contidos no interior de i e os arcos contidos no interior de j. Para cada arco r em um desses conjuntos, somamos ou subtraímos l_r de alguns elementos de A e B, de forma que as matrizes passem a corresponder à nova ordem dos discos.

A realização de alguns experimentos mostrou que a maioria das trocas que são executadas envolvem discos em níveis próximos. No entanto, a qualidade das soluções encontradas diminui se as trocas entre discos em níveis distantes são ignoradas. Por isso, a busca com essa vizinhança começa explorando trocas entre discos em níveis consecutivos e aumenta progressivamente a diferença de nível entre os discos. Sempre que é encontrada uma troca $T(i,j)$ que gera alguma melhoria, ela é executada e a busca é reiniciada. Porém, nem todos os movimentos precisam ser avaliados novamente. Dados dois discos i' e j', suponha que $|x^{-1}(i') - x^{-1}(j')| < |x^{-1}(i) - x^{-1}(j)|$. Além disso, suponha que i' e j' estejam ambos abaixo ou acima de i e j. Então, a variação no valor da solução causada por $T(i',j')$ é a mesma antes e depois da execução de $T(i,j)$. Essa observação é utilizada para evitar que movimentos sejam reavaliados sem necessidade.

4 Path-relinking

Path-relinking é aplicado a todos os máximos locais produzidos pela busca local. Três estratégias existentes na literatura foram consideradas: *forward*, *backward* e *mixed path-relinking* (Resende e Ribeiro (2009) fornecem uma descrição dessas estratégias). Um *conjunto de elite* é utilizado para armazenar as N_{elite} melhores soluções encontradas durante a execução da heurística. Após a busca local, uma das soluções desse conjunto é selecionada aleatoriamente e *path-relinking* é aplicado entre ela e o máximo local, ou seja, uma seqüência de movimentos é aplicada a uma das soluções para transformá-la na outra. Para selecionar a solução de elite, atribui-se à cada solução x_e do conjunto de elite uma probabilidade p_e de ser escolhida proporcional à distância de x_e ao máximo local. A melhor solução intermediária visitada pode ser submetida à busca local, já que ela não é necessariamente um máximo local. Em nossa implementação, restringimos a aplicação da busca local aos casos em que a melhor solução intermediária é melhor que as duas soluções às quais está sendo aplicado *path-relinking*. As soluções resultantes desse processo são candidatas a inclusão no conjunto de elite seguindo as regras descritas por Resende e Ribeiro (2009).

Para medir a distância entre duas soluções, consideramos três métricas: o menor número de inserções (*métrica de inserção*) e o menor número de trocas (*métrica de troca*) necessários para transformar uma solução na outra, e o número de inversões entre as soluções (*métrica de inversão*). Dados dois discos i e j e duas soluções x_1 e x_2, dizemos que uma inversão ocorre quando $x_1^{-1}(i) > x_1^{-1}(j)$ e $x_2^{-1}(i) < x_2^{-1}(j)$. Executamos movimentos de inserção com as métricas de inserção e de inversão e movimentos de troca com a métrica de troca. Pode-se mostrar que as métricas de inserção e de troca são equivalentes às distâncias de Ulam e de Cayley, respectivamente (veja Diaconis (1988)). Além disso, $O(n)$ movimentos são executados com essas métricas, enquanto

que com a métrica de inversão $O(n^2)$ movimentos são necessários. Por isso, esta última consome mais tempo, mas explora um número maior de soluções e geralmente obtém resultados melhores.

Dizemos que um movimento é *válido* durante o *path-relinking* se sua execução diminui a distância entre as soluções consideradas. A cada iteração do *path-relinking*, o melhor movimento válido é selecionado e executado. O número de movimentos válidos a cada iteração sob as métricas de inserção e de troca é relativamente pequeno. Porém, com a métrica de inversão podem haver muitos movimentos válidos e avaliar todos eles pode consumir muito tempo. Por isso, algumas informações são mantidas entre as iterações do *path-relinking* para acelerar essa avaliação.

Seja x a solução corrente no *path-relinking* com a métrica de inversão. Suponha que a última inserção executada foi $I(i,p)$ e que antes da execução dessa inserção i estava no nível q, com $p > q$. Dizemos que um disco i' é afetado por $I(i,p)$ quando $q \leq x^{-1}(i') \leq p$. Seja j um disco que não foi afetado por $I(i,p)$. Denotamos por $M_j^{<q}$ e $M_j^{>p}$ os conjuntos de inserções de j em níveis abaixo de q e acima de p, respectivamente. Considere a seleção do próximo movimento a ser executado. Os melhores movimentos em $M_j^{<q}$ e em $M_j^{>p}$ ainda são os mesmos de antes da execução de $I(i,p)$. Além disso, um movimento em $M_j^{<q} \bigcup M_j^{>p}$ é válido após a execução de $I(i,p)$ se, e somente se, ele também era válido antes. Portanto, para encontrar a melhor inserção de j, basta comparar as inserções entre os níveis p e q com as melhores em $M_j^{<q}$ e em $M_j^{>p}$.

Para trabalhar com esses conjuntos, foi criada uma árvore binária balanceada L_i para cada disco $i \in S$. Cada árvore L_i tem n folhas, as quais armazenam um inteiro que representa um nível no qual i pode ser inserido. Quando $I(i,p)$ é inválido, a folha correspondente ao nível p também recebe a marcação. Dado um nó v, denotamos por P_v o conjunto de níveis que estão representados nas folhas da subárvore com raiz em v. Cada nó interno v armazena um nível $p_v \in P_v$, tal que $I(i,p_v)$ é a melhor inserção válida, considerando-se os níveis em P_v. Quando todas as folhas da subárvore com raiz em v estão marcadas como inválidas, v também é marcado. A melhor inserção para o disco i é encontrada examinando-se a raiz de L_i. Um exemplo pode ser visto na Figura 3.

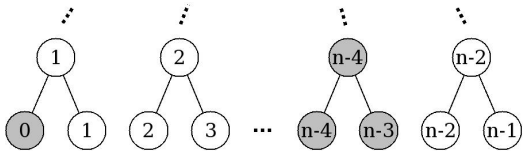

Figura 3: Árvore utilizada durante o *path-relinking*. Nós inválidos são mostrados em cinza.

Cada árvore L_i é construída em tempo $O(n)$. Uma atualização é necessária em dois casos: quando o valor de uma inserção do disco i é alterado e quando há uma mudança na validade de uma inserção. A atualização das árvores é iniciada nas folhas e propagada até a raiz. Em uma folha, deve-se apenas checar a validade do movimento. Em um nó interno v, sejam p e q os níveis armazenados nos filhos de v. Se $I(i,p)$ é melhor do que $I(i,q)$, v armazena o nível p, caso contrário ele armazena o nível q. Seja k o número de discos afetados pelo último movimento executado. Quando i é um disco que não foi afetado pelo último movimento executado, k folhas devem ser atualizadas. As alterações devem ser propagadas até a raiz de L_i e o número de nós visitados é $O(k + \log n)$. Se i foi afetado pelo último movimento, é possível que $O(n)$ nós sejam visitados devido a mudanças na validade das inserções de i. Assim, o tempo gasto para atualizar todas as árvores após a execução de um movimento é $O(kn + (n-k)(k + \log n)) = O(kn + n \log n)$. A melhor inserção válida é encontrada comparando-se as melhores inserções de todos os discos, o que pode ser feito em tempo linear. Experimentos preliminares mostraram que k independe de n e que seu valor é menor que 6 para a maioria das instâncias consideradas. Dessa forma, o método apresentado produz uma redução significativa dos tempos de execução do *path-relinking*.

Evolutionary path-relinking é uma estratégia que aplica *path-relinking* a pares de soluções do conjunto de elite. Neste projeto, ela é aplicada após o fim das iterações do GRASP. Resende e Ribeiro (2009) descrevem dois métodos existentes na literatura. Nós propomos uma variação desses

métodos, na qual associamos a cada solução de elite x uma solução x' obtida revertendo-se a ordem dos discos em x. Assim, a distância entre x e x' é máxima sob a métrica de inversão e um grande número de soluções intermediárias pode ser visitado. *Path-relinking* é aplicado a x e x' e a melhor solução intermediária é submetida à busca local, caso ela seja melhor que x e x'. Esse procedimento é repetido com a solução resultante até que nenhuma melhoria seja obtida. Nos experimentos realizados, esse método encontrou soluções melhores do que os demais.

5 Resultados computacionais

Inicialmente, realizamos experimentos preliminares para selecionar as configurações do GRASP que produzem as soluções de mais alta qualidade. Em todos os experimentos, executamos 1000 iterações do GRASP. Com relação à fase de construção, alguns testes foram feitos com os valores $\alpha = 0.0, \alpha = 0.1, \ldots, \alpha = 1.0$ fixos durante todas as iterações e os melhores resultados foram obtidos com $\alpha = 0.4$. Dentre as estratégias que utilizam valores variáveis de α, o método *GRASP reativo* foi o que apresentou melhor desempenho (Resende e Ribeiro (2009) fornecem uma descrição do método). A qualidade das soluções produzidas com GRASP reativo foi muito semelhante a das soluções encontradas fixando-se $\alpha = 0.4$, mas os tempos de execução foram maiores. Por isso, nos experimentos subseqüentes, utilizamos $\alpha = 0.4$. Também foram realizados experimentos com funções de viés e os melhores resultados foram obtidos com viés aleatório (*random bias*), isto é, atribui-se a todos os elementos da LRC a mesma probabilidade de serem escolhidos.

Os experimentos também indicaram que a busca local com a vizinhança de inserção é executada em menos tempo e encontra soluções de maior qualidade do que com a vizinhança de troca. O método VND implementado teve tempos de execução comparáveis aos da busca local com a vizinhança de inserção e, para um grande número de instâncias, encontrou soluções melhores do que as que foram obtidas pela aplicação de apenas uma das vizinhanças. Por isso, ele foi utilizado na fase de busca local. Além disso, os resultados mostraram que o *path-relinking* encontra melhores soluções com a estratégia *mixed path-relinking* implementada com a métrica de inversão e um conjunto de elite de tamanho $N_{elite} = 10$.

Com essas configurações, realizamos experimentos com dois conjuntos de instâncias, os quais denotamos por C_1 e C_2. O conjunto C_1 foi utilizado no trabalho de Cabello et al. (2010) e é composto por 4 instâncias. O conjunto C_2 é formado por 28 instâncias geradas a partir de dados sobre a população de cidades em diversos países. Em todos os experimentos, foram aplicadas as técnicas de decomposição descritas por Kunigami et al. (2010) e os resultados do GRASP são comparados aos do método exato descrito naquele trabalho. A máquina utilizada tem processador Intel Core 2 Quad 2.83GHz, com 8GB de memória RAM. A heurística foi implementada em C++ e compilada com GCC 4.4.4. Os resultados obtidos são mostrados nas tabelas 1 e 2. Os valores apresentados descontam os comprimentos dos arcos que estão visíveis em todas as soluções. Os casos em que a heurística encontrou uma solução ótima são destacados em negrito. A coluna %Ótimo mostra a diferença relativa entre o valor encontrado pelo GRASP e o valor ótimo, em porcentagens. Os tempos de execução são dados em segundos. A coluna T.Exato/T.GRASP indica quantas vezes a heurística foi mais rápida do que o método exato.

A Tabela 1 mostra os resultados dos experimentos realizados com as instâncias de C_1. Conforme observado por Kunigami et al. (2010), a instância *City 156* pode ser decomposta em componentes relativamente pequenas e, por isso, ela não apresentou dificuldades para nenhum dos dois métodos. O tempo registrado é praticamente todo utilizado na decomposição da instância original. A heurística não encontrou uma solução ótima para a instância *Earth. death*, porém a diferença relativa entre o valor encontrado e o ótimo foi menor que 0.01%. Para as outras duas instâncias, o tempo de execução da heurística foi da ordem de centenas de vezes menor que o do método exato e, no caso em que uma solução ótima não foi encontrada, o valor obtido foi apenas 0.22% menor que o da ótima.

A Tabela 2 mostra alguns resultados dos experimentos realizados com as instâncias de C_2.

Instâncias	n	Valor GRASP	Valor Exato	%Ótimo	Tempo GRASP	Tempo Exato	T.Exato/ T.GRASP
City 156	156	**541.29**	541.29	0.00	3	4	1.3
City 538	538	**503.27**	503.27	0.00	34	31067	913.7
Earth. mag.	602	12147.25	12174.09	0.22	91	52581	577.8
Earth. death	602	2915.09	2915.11	0.00	13	50	3.8

Tabela 1: Resultados obtidos com o conjunto de instâncias C_1.

Soluções ótimas foram obtidas em 15 das 28 instâncias consideradas. Além disso, nos casos em que não foi encontrada uma solução ótima, o valor da melhor solução fornecida pelo GRASP foi, em média, 0.07% menor que o ótimo, sendo que a maior diferença relativa foi de 0.23%. O tempo de execução da heurística foi, em média, 110.3 vezes menor que o do método exato (mínimo = 13.6 vezes, máximo = 245.7 vezes). O mapa à direita da Figura 1 corresponde à solução fornecida pelo GRASP para componentes formadas por 25 discos da instância *Brasil*.

Instâncias	n	Valor GRASP	Valor Exato	%Ótimo	Tempo GRASP	Tempo Exato	T.Exato/ T.GRASP
China	141	**2409.15**	2409.15	0.00	20	2882	144.1
Brasil	150	**2553.52**	2553.52	0.00	11	982	89.3
Rússia	150	1565.51	1566.56	0.07	10	1003	100.3
Austrália	210	**2115.59**	2115.59	0.00	16	2782	173.9
Dinamarca	310	2295.91	2301.09	0.23	13	1314	101.1
Holanda	372	4719.09	4720.45	0.03	28	4175	149.1

Tabela 2: Resultados obtidos com o conjunto de instâncias C_2.

6 Conclusão

Foi apresentada uma heurística sofisticada baseada em GRASP com *path-relinking* para o problema Max-Total para empilhamentos. Os resultados experimentais apresentados mostram que a heurística encontra soluções ótimas para um grande número de instâncias geradas a partir de dados reais. Quando isso não ocorre, a diferença relativa entre o valor da melhor solução fornecida e o ótimo é menor do que 0.23%. Além disso, os tempos de execução são da ordem de centenas de vezes menores que os do método exato descrito na literatura.

Agradecimentos: os autores agradecem à FAPESP (pr. 07/52015-0 e 2009/17044-5), ao CNPq (pr. 483177/2009-1, 473867/2010-9 e 302804/2010-2) e ao FAEPEX/UNICAMP pelo apoio financeiro.

Referências

Cabello, S., Haverkort, H., van Kreveld, M., e Speckmann, B. (2010). Algorithmic aspects of proportional symbol maps. *Algorithmica*, 58(3), 543–565.

de Berg, M., Cheong, O., van Kreveld, M., e Overmars, M. (2008). *Computational Geometry: Algorithms and Applications*. Springer-Verlag, terceira edição.

Diaconis, P. (1988). *Group Representations in Probability and Statistics*. Institute of Mathematical Statistics.

Glover, F. (1996). Tabu search and adaptive memory programing – advances, applications and challenges. Em Barr, R., Helgason, R., e Kennington, J. (Eds.), *Interfaces in Computer Science and Operations Research*, 1–75. Kluwer Academic Publishers.

Kunigami, G., de Rezende, P. J., de Souza, C. C., e Yunes, T. (2010). Optimizing the layout of proportional symbol maps. *Optimization Online*.

Laguna, M. e Martí, R. (1999). GRASP and path relinking for 2-layer straight line crossing minimization. *INFORMS Journal on Computing*, 11, 44–52.

Resende, M. G. C. e Ribeiro, C. C. (2009). Greedy randomized adaptive search procedures: Advances, hybridizations, and applications. Em Glover, F. e Kochenberger, G. A. (Eds.), *Handbook of Metaheuristics*, 219–249. Springer, segunda edição.