Um GRASP com reconexão de caminho para o *p*- problema mediano *

Mauricio GC Resende † Renato F. Werneck ‡

Abstrato

Dado *n* clientes e um conjunto *F* de *m* instalações potenciais, o *p*- problema mediano consiste em encontrar um subconjunto de *F* com *p* instalações de tal forma que o custo de atendimento a todos os clientes seja minimizado. Este é um problema NPcompleto bem conhecido com aplicações importantes em ciência de localização e classificação (clustering). Apresentamos aqui um GRASP (Greedy Randomized Adaptive Search Procedure) com redirecionamento de caminhos para encontrar soluções quase ótimas para esse problema. Resultados empíricos em instâncias da literatura sugerem que este é um algoritmo muito robusto, com desempenho pelo menos tão bom quanto outros métodos, e muitas vezes melhor em termos de tempo de execução e qualidade da solução.

1. Introdução

O *p*- o problema médio é definido como segue. Dado um conjunto *F* de *m* instalações potenciais, um conjunto *você* de *n* usuários (ou clientes), uma função à distância $d: U \times F \rightarrow R$, e uma constante $p \leq m$, determinar qual *p* instalações a serem abertas de forma a minimizar a soma das distâncias de cada usuário até a instalação aberta mais próxima.

Vários algoritmos para o *p*- problemas de mediana foram propostos, incluindo métodos exatos baseados em programação linear [2, 3, 8, 30], algoritmos construtivos [3, 17, 38], algoritmos de base dupla [8, 21] e procedimentos de busca local [13, 16, 19, 24, 27, 35, 38]. Mais recentemente, metaheurísticas capazes de obter soluções de qualidade quase ótima foram desenvolvidas. Procedimentos de busca tabu foram propostos por Voss [37] e Rolland et al. [26]. O último método foi comparado em [29] com Rosing e ReVelle's *concentração heurística* método [28], que obteve resultados comparativamente superiores. Em [14], Hansen e Mladenović´ propõem um VNS (busca de vizinhança variável) para o problema, posteriormente paralelizado por García-López et al. em [10]. Uma variação deste método, VNDS (pesquisa de decomposição de vizinhança variável), foi proposta por Hansen, Mladenović´ e Perez-Brito [15]. Heurísticas baseadas em programação linear foram propostas por du Merle et al. [4] e por Senne e Lorena [32, 33].

Neste artigo, propomos um GRASP híbrido (Greedy Randomized Adaptive Search Procedure) para o *p*- problema mediano. GRASP é uma metaheurística iterativa multistart randomizada [5, 6, 7, 23], na qual cada iteração consiste em uma fase construtiva (quando um algoritmo guloso randomizado é executado) seguida por uma fase de busca local. A melhor solução obtida em todas as iterações é considerada o resultado de todo o algoritmo.

Nosso algoritmo aprimora essa abordagem básica com algumas estratégias de intensificação. Mantemos um pool com algumas das melhores soluções encontradas em iterações anteriores, as chamadas *soluções de elite.* Em cada iteração do GRASP, a solução obtida após a fase de busca local é combinada com uma solução de elite por meio de um processo denominado *revinculação de caminhos.* Além disso, depois que todas as iterações são concluídas, temos uma segunda fase de pós-otimização, na qual as soluções de elite são combinadas entre si. A Figura 1 mostra o pseudocódigo para nossa estratégia geral.

* Relatório técnico de pesquisa do AT&T Labs TD-5E53XL, 18 de setembro de 2002.

† AT&T Labs Research, Florham Park, NJ 07932. E-mail: mgcr@research.att.com.

‡ Princeton University, Princeton, NJ 08544. E-mail: rwerneck@cs.princeton.edu. Os resultados apresentados neste artigo foram obtidos enquanto este autor era um estagiário de verão na AT&T Labs Research.

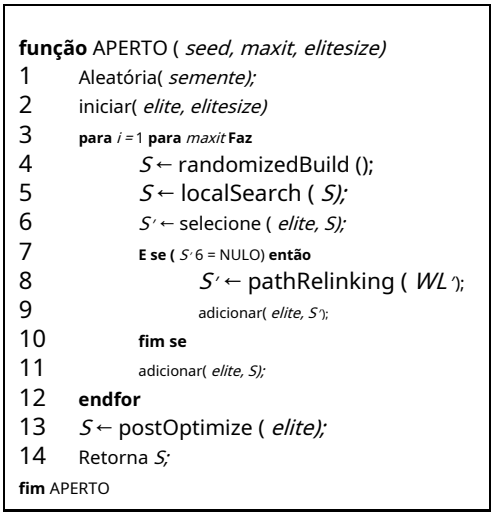

Figura 1: Pseudo-código para o procedimento GRASP

Claro, ainda há muito a ser especificado para transformar esse esboço em um algoritmo real. Estudamos cada componente individual (algoritmo construtivo, busca local e intensificação) separadamente nas Seções 3, 4 e 5, respectivamente. Na seção 6, apresentamos os resultados obtidos pela versão final de nosso método e os comparamos com aqueles obtidos por outros métodos da literatura. Mas, primeiro, na Seção 2, discutiremos questões importantes relacionadas aos experimentos que conduzimos tanto para avaliar componentes individuais quanto para produzir os resultados finais.

2 testes

2.1 Instâncias

Testamos nosso algoritmo em cinco classes de problemas: TSP, ORLIB, SL, GR, e RW.

Instâncias em aula TSP são apenas conjuntos de pontos no plano. Propostos originalmente para o problema do caixeiro viajante, estão disponíveis no TSPLIB [22]. No contexto do *p*- problema médio, essas instâncias foram usadas pela primeira vez por Hansen et al. [14, 15]. Cada ponto é considerado uma instalação potencial e um cliente, e o custo de atribuição do cliente *c* para facilidade *f* é simplesmente a distância euclidiana entre os pontos que representam *c* e *f*. Como em [15], três instâncias são consideradas (fl1400, pcb3038, e rl5934, com 1400, 3038 e 5934 nós, respectivamente), cada um com vários valores diferentes para *p (* número de instalações a abrir).

Aula ORLIB (abreviação de OR-Library) foi introduzida por Beasley em [2]. Cada uma das 40 instâncias (pmed01, pmed02,. . ., pmed40) na classe é um gráfico com um valor correspondente para *p.* Cada nó é um cliente e uma instalação em potencial, e o custo de atribuir um cliente a uma instalação é o comprimento do caminho mais curto entre os nós correspondentes. O número de nós nesta classe varia de 100 a 900, e o valor de *p* de 5 a 200.

A terceira classe que consideramos é SL, uma ligeira extensão para ORLIB proposto por Senne e Lorena em [32]. Ele contém três novas instâncias, todas baseadas em gráficos de ORLIB: sl700 usa o mesmo gráfico que pmed34, mas usa $p = 233$; sl800 é o mesmo que pmed37, com $p = 267$; e sl900 é pmed40 com $p = 300$ [31].

A quarta aula estudada é GR, introduzido por Galvão e ReVelle [9] e usado para o *p*- problema da mediana por Senne e Lorena [32]. Esta classe contém dois gráficos, com 100 e 150 nós (nomeados gr100 e gr150, respectivamente). Oito valores de *p (* entre 5 e 50) foram considerados em cada caso.

A quinta aula que estudamos, RW, foi originalmente usado em [24] e corresponde a uma distância completamente aleatória

matrizes. A distância entre cada instalação e cada cliente tem um valor inteiro tomado uniformemente ao acaso a partir do intervalo [1, *n],* Onde *n* é o número de clientes. [1] Em todos os casos, o número de instalações potenciais (*m)* é igual ao número de clientes (*n).* Quatro valores diferentes de *n* foram considerados: 100, 250, 500 e 1000 (os nomes das instâncias são rw100, rw250, rw500, e rw1000, respectivamente). Em cada caso, vários valores diferentes de *p* foram testados. O programa que criou essas instâncias, que usa o gerador de números aleatórios de Matsumoto e Nishimura [20], está disponível aos autores mediante solicitação.

Os custos são essenciais em todas as classes, exceto TSP, em que as distâncias são, em teoria, valores reais. Não arredondamos nem truncamos explicitamente os valores, que foram mantidos com Duplo precisão em todo o algoritmo.

Os resultados obtidos pela versão final do nosso algoritmo em todas as instâncias são mostrados na Seção 6. Também conduzimos experimentos com várias variações do algoritmo para avaliar como cada componente individual (algoritmo construtivo, reconexão de caminho, busca local, entre outros) afeta o desempenho geral do método. Nesses experimentos, no entanto, usamos apenas um *conjunto restrito* de instâncias. Este conjunto foi escolhido com dois objetivos em mente. Primeiro, suas instâncias devem ser difíceis o suficiente para tornar evidentes as diferenças entre vários parâmetros e componentes. Alguns casos, especialmente em sala de aula ORLIB, podem ser resolvidos até a otimização apenas pela pesquisa local, tornando inútil incluí-los nesses experimentos. Nosso segundo objetivo era manter o conjunto pequeno o suficiente para permitir experimentos estatisticamente significativos (ou seja, com várias sementes aleatórias para cada instância) em uma quantidade relativamente pequena de tempo de CPU (não mais do que alguns dias por experimento). Dados esses objetivos e nossa experiência com as primeiras versões de nosso algoritmo, definimos o conjunto restrito com 10 instâncias: pmed15 e pmed40, ambos da aula ORLIB; sl700, da aula SL; fl 1400 (com $p = 150$ e $p = 500$) e pcb3038 (com $p = 30$ e $p = 250$), da aula TSP; gr150 (com $p = 25$), da aula GR; e rw500 (com $p = 25$ e $p = 75$), da aula RW.

2.2 Ambiente de Teste

Os testes foram realizados em um desafio SGI com 28 processadores MIPS R10000 de 196 MHz (com cada execução do programa limitada a apenas um processador) e 7,6 GB de memória. O algoritmo foi codificado em C ++ e compilado com o compilador SGI MIPSpro C ++ (v. 7.30) com fl ags - O3-OPT: Olimit = 6586. Todos os tempos de execução mostrados neste documento são tempos de CPU, medidos com o getrusage função. Salvo indicação em contrário, os tempos de execução não incluem o tempo gasto lendo instâncias do disco (em particular, nas aulas SL e ORLIB, eles incluem o tempo necessário para calcular os caminhos mais curtos de todos os pares a partir da leitura do gráfico). O gerador de números aleatórios que usamos é o de Matsumoto e Nishimura *Mersenne Twister [* 20].

3 algoritmos construtivos

O algoritmo ganancioso padrão para o *p*- o problema mediano [3, 38] começa com uma solução vazia e adiciona instalações, uma de cada vez; em cada iteração, ele escolhe a facilidade mais lucrativa, ou seja, aquela cuja adição causa a maior queda no custo da solução. Evidentemente, esse método não pode ser usado diretamente no framework GRASP: sendo completamente determinístico, ele produziria soluções idênticas em todas as iterações. Nós consideramos as seguintes variações aleatórias em nossos experimentos:

- aleatória (solução aleatória): Selecione *p* instalações uniformemente ao acaso. A própria seleção requer *O (m)* tempo, e determinar qual instalação deve atender cada cliente exige *O (pn)* operações. [2] Portanto, a complexidade geral do algoritmo é *O (m + pn).*
- rpg (aleatório mais ganancioso): Selecione uma fração α (um parâmetro de entrada) do *p* instalações aleatoriamente e, em seguida, conclua a solução de forma gananciosa. O pior caso de complexidade do algoritmo é *O ((* 1 - α) (*pmn) +* α (*m + pn)),* que corresponde a *O (pmn)* E se α não é muito próximo de 1. Em nossos testes, um valor para α foi escolhido uniformemente ao acaso no intervalo [0; 1] em cada iteração GRASP.

[1] Em particular, ao contrário de todas as outras séries, a distância da instalação *eu* ao usuário *eu* não é zero. Além disso, a distância entre as instalações *eu* e usuário *j* não precisa ser igual à distância entre as instalações *j* e usuário *eu.*

[2] Isso pode ser feito mais rápido em algumas configurações, como gráficos esparsos ou pontos no plano euclidiano. Os resultados neste artigo, no entanto, não usam nenhuma dessas acelerações métricas específicas.

- rgreedy (ganância aleatória): Semelhante ao algoritmo ganancioso, mas em cada etapa *eu,* em vez de selecionar o melhor entre todos *m* - *i* + 1 opção, escolha aleatoriamente entre d α (*m* - *i* + 1) e melhores opções, onde 0 < α ≤ 1 é um parâmetro de entrada. Observe que se α → 0, este método degenera no algoritmo ganancioso; E se α → 1, ele se transforma no algoritmo aleatório. Em nossos testes, selecionamos α uniformemente ao acaso no intervalo (0; 1] em cada iteração GRASP. Este algoritmo leva *O (pmn)* Tempo.

- pgreedy (ganância proporcional): Mais uma variação do algoritmo ganancioso. Em cada iteração, com pute, para cada instalação candidata $f_{eu,}$ quanto seria economizado se f_{eu} foram adicionados à solução. Deixar $s(f_{eu})$ seja este valor. Em seguida, escolha um candidato ao acaso, mas de forma tendenciosa: a probabilidade de um determinado instalação f_{eu} sendo selecionado é proporcional a $s(f_{eu}) - \min_j s(f_j)$. Se todos os candidatos forem igualmente bons (todos economizariam a mesma quantia), selecione um uniformemente ao acaso. Este método também leva *O (pmn)* Tempo.

- pworst ((pior proporcional): Neste método, a primeira instalação é selecionada uniformemente ao acaso. Outras instalações são adicionadas uma de cada vez da seguinte forma. Calcule, para cada cliente, a diferença entre quanto custaria sua atribuição atual e quanto custaria a melhor atribuição; em seguida, selecione um cliente aleatoriamente, com probabilidade proporcional a esse valor, e abra a unidade mais próxima. Os clientes para os quais a solução atual é particularmente ruim têm maior chance de serem selecionados. Este método, também usado em [34], é executado em *O (mn)* Tempo.

- amostra (amostra gulosa): Este método é semelhante ao algoritmo guloso, mas em vez de selecionar a melhor entre todas as opções possíveis, ele apenas considera *q* <*m* possíveis inserções (escolhidas uniformemente ao acaso) em cada iteração. O mais lucrativo entre eles é selecionado. O tempo de execução do algoritmo é *O (m + pqn).* A ideia é fazer *q* pequeno o suficiente para reduzir significativamente o tempo de execução do algoritmo (quando comparado ao guloso puro) e para garantir um grau razoável de randomização. Dentro nossos testes, usamos *q* = d registro 2 (*m* / *k)* e.

Não estava claro a princípio qual desses métodos seria mais adequado como um bloco de construção de nossa heurística. Para analisar melhor essa questão, conduzimos um experimento no conjunto restrito de instâncias (definido na Seção 2.1). Para cada instância no conjunto e cada procedimento construtivo, rodamos nossa heurística 10 vezes, com 10 sementes aleatórias diferentes. Em todos os casos, executamos 32 iterações GRASP, com 10 soluções de elite, usando cima baixo como o critério para determinar a direção da reconexão do caminho (esse critério é definido na Seção 5.4.2).

Para explicar os resultados mostrados na Tabela 1, precisamos de algumas definições. Para cada instância *EU,* nós computamos *média (I),* o valor médio da solução (obtido por todo o procedimento GRASP com pós-otimização) considerando todas as 60 execuções (6 métodos diferentes, cada um com 10 sementes aleatórias). Então, para cada método, determinamos o *desvio percentual relativo* para essa instância: quanto o valor médio da solução obtido por esse método está acima (ou abaixo) *média (I),* em termos percentuais. Tomando a média desses desvios em todas as 10 instâncias, obtemos o *desvio percentual médio relativo* para cada método; esses são os valores mostrados na coluna 2 da Tabela 1. A coluna 4 foi calculada de maneira semelhante, mas considerando os tempos de execução em vez dos valores da solução.

As colunas 3 e 5 foram calculadas como segue. Para cada instância, os métodos foram classificados de acordo com seus desvios percentuais relativos; o melhor recebeu um ponto, o segundo dois pontos e assim sucessivamente, até o sexto melhor método, com seis pontos. Quando havia empate, os pontos eram divididos igualmente entre os métodos envolvidos. Por exemplo, se os desvios foram - 0,03, - 0,02, - 0,02, 0,01, 0,03 e 0,03, os métodos receberiam 1,2,5,2,5,4,5,5 e 5,5 pontos, respectivamente. O número de pontos recebidos por um método de acordo com este processo é o seu *classificação* para essa instância específica. Seu *classificação normalizada* foi obtido mapeando linearmente o intervalo de classificações (1 a 6, neste caso) para o intervalo [- 1,1]. No exemplo acima, as classificações normalizadas seriam - 1, - 0,4, - 0.4,0.2.0.8 e 0.8. O normalizado deve somar zero (por definição). Se um método é sempre melhor do que todos os outros, sua classificação normalizada será - 1; se sempre pior, será 1. O que as colunas 3 e 5 da Tabela 1 mostram são os *classificações normalizadas médias* de cada método, assumido o conjunto de 10 instâncias. A coluna 3 refere-se à qualidade da solução e a coluna 5 aos tempos de execução.

A correlação entre essas medidas é maior quando um método é obviamente melhor (ou pior) do que outros métodos. Em geral, no entanto, ter uma classificação normalizada média inferior não significa ter uma melhor

Tabela 1: Resultados médios GRASP com diferentes procedimentos construtivos: Desvios percentuais relativos médios (% DEV) e classificações normalizadas médias (NRANK) para qualidades de solução e tempos de execução (ambos se referindo a todo o procedimento GRASP com pós-otimização). Valores menores são melhores.

MÉTODO	QUALIDADE		TEMPO	
	DEV%	NRANK	DEV%	NRANK
pgreedy	- 0,009	0,160	39,6	0,920
pworst	- 0,006	- 0,400	- 18,7	- 0,480
rgreedy	0,020	- 0,160	35,8	0,400
aleatória	0,015	0,000	- 24,9	- 0,840
rpg	0,009	0,620	- 12,3	- 0,300
amostra	- 0,029	- 0,220	- 19,5	- 0,600

desvio percentual médio relativo, como mostra a tabela.

A tabela deixa claro que os métodos se distinguem muito mais pelo tempo que demoram do que pela qualidade das soluções que fornecem. Como sugere a análise de suas complexidades de pior caso, rgreedy e pgreedy são muito mais lentos do que os outros métodos. Na verdade, eles são tão mais lentos que, conforme mostrado na tabela, fazem com que toda a heurística GRASP demore em média duas vezes mais do que quando se usam métodos mais rápidos. O outro método com *O (pmn)* desempenho de pior caso, rpg, é muito mais rápido que rgreedy e pgreedy na prática, mas ainda é mais lento do que outros métodos, sem encontrar melhores soluções em média. Portanto, tendemos a favorecer os três métodos relativamente rápidos: pworst, amostra, e aleatória. Entre aqueles, amostra e pworst parecem ser capazes de encontrar soluções de qualidade ligeiramente melhor. Nós escolhemos amostra para a versão final do nosso algoritmo, embora pworst provavelmente encontraria resultados muito semelhantes.

Não se pode deixar de notar que este experimento revela uma característica incomum do *p*- problema mediano. Geralmente, na estrutura GRASP, o tempo de execução da heurística construtiva não é um problema. Normalmente, é uma boa ideia usar métodos construtivos aleatórios que produzam uma variedade de soluções de alta qualidade, uma vez que isso reduziria o número de iterações do procedimento de pesquisa local geralmente muito mais lento. Em nosso caso, a busca local é tão rápida (e os algoritmos construtivos relativamente tão lentos), que investir tempo extra construindo a solução pode realmente tornar o GRASP muito mais lento sem nenhum ganho significativo em termos de qualidade da solução. Não poderíamos usar a estratégia de randomização normalmente usada no GRASP, representada aqui por rgreedy;
em vez disso, tivemos que desenvolver uma alternativa mais rápida com base na amostragem.

4 Pesquisa Local

O procedimento padrão de pesquisa local para o *p*- O problema da mediana, originalmente proposto por Teitz e Bart [35] e estudado ou usado por vários autores [10, 14, 15, 16, 24, 38], é baseado em facilidades de troca. Dada uma solução inicial *S,* o procedimento determina, para cada instalação *f* 6 ∈ *S,* qual facilidade *g* ∈ *S (* se houver) melhoraria mais a solução se *f* e *g* foram trocados (ou seja, se *f* foram abertos e *g* fechado). Se houver um movimento de "melhoria", *f* e *g* são trocados. O procedimento continua até que nenhuma troca de melhoria possa ser feita, caso em que um *mínimo local* terá sido encontrado.

Em [38], Whitaker descreve uma implementação eficiente deste método, que ele chamou *intercâmbio rápido.* Uma implementação semelhante é usada por Hansen e Mladenović´ em [14] (e posteriormente, por vários autores, em [10] e [15]). A única pequena diferença entre eles é o fato de que Whitaker prefere um *primeira melhoria* estratégia (o algoritmo muda para uma solução vizinha, logo que encontra uma melhoria), enquanto nos outros casos a estratégia preferida é *melhor melhoria (* todos os vizinhos são verificados e o melhor é escolhido). Em qualquer caso, o tempo de execução de cada iteração é limitado por *O (mn).*

Em [24], Resende e Werneck propõem uma implementação alternativa para o procedimento de busca local. Embora tenha a mesma complexidade do pior caso que o de Whitaker, pode ser substancialmente mais rápido na prática. A aceleração (de até três ordens de magnitude) resulta do uso de informações coletadas nas primeiras iterações do algoritmo para reduzir a quantidade de computação realizada em estágios posteriores. Isso, no entanto, requer um

maior quantidade de memória. Embora a implementação de Whitaker exija *Sobre)* memória no pior caso (não considerando a matriz de distância), a implementação em [24] pode usar até *O (mn)* posições de memória.

Em qualquer caso, acreditamos que a aceleração vale bem a pena o requisito de memória extra. Isso é especialmente verdadeiro para métodos que dependem fortemente de procedimentos de busca local, que incluem não apenas o método descrito aqui (GRASP), mas também VNS [14] e busca tabu [26, 37], por exemplo. Além disso, deve-se também lembrar que, embora a memória extra seja assintoticamente relevante quando a função de distância é dada implicitamente (como no caso de instâncias euclidianas), é irrelevante quando há um real *O (mn)* matriz de distância (como em série RW). Diante dessas considerações, optou-se por utilizar neste artigo a versão mais rápida descrita em [24], que requer Θ (*mn)* posições de memória.

Visto que dificilmente é trivial, nos abstivemos de descrever a implementação aqui. O leitor é consultado em [24] para detalhes e para uma comparação experimental com a implementação de Whitaker.

5 Intensificação

Nesta seção, descrevemos os aspectos de intensificação de nossa heurística. Funciona através de uma piscina de *soluções de elite,* soluções de alta qualidade encontradas durante a execução. A intensificação ocorre em dois estágios diferentes, como mostra a Figura 1. Primeiro, cada iteração GRASP contém uma etapa de intensificação, na qual a solução recém-gerada é combinada com uma solução do pool. Em seguida, na fase de pós-otimização, as soluções do pool são combinadas entre si. Em ambas as fases, a estratégia usada para combinar um par de soluções é a mesma: *revinculação de caminhos.* Originalmente proposto para tabu search e scatter search [11, 12], este procedimento foi aplicado pela primeira vez dentro do framework GRASP por Laguna e Martí [18], e amplamente aplicado desde então (ver [23] para alguns exemplos). A subseção 5.1 descreve brevemente como funciona a reconexão de caminhos. A Subseção 5.2 explica as regras pelas quais o pool é atualizado e as soluções são obtidas a partir dele. Finalmente, a Subseção 5.3 descreve a fase de pós-otimização.

5.1 Revinculação de caminho

Deixar S_1 e S_2 ser duas soluções válidas, interpretadas como conjuntos de instalações (abertas). O procedimento de religação de caminho começa com uma das soluções (digamos, S_1) e gradualmente o transforma no outro (S_2) trocando elementos que estão em $S_2 \setminus S_1$ e trocar elementos que estão em $S_1 \setminus S_2$ O número total de trocas realizadas é $| S_2 \setminus S_1 |$, que é igual a $| S_1 \setminus S_2 |$; este valor é conhecido como o *diferença simétrica* entre S_1 e S_2 A escolha de qual troca fazer em cada estágio é ambiciosa: sempre realizamos a jogada mais lucrativa (ou menos custosa).

Conforme mencionado em [23], o resultado do método é geralmente a melhor solução encontrada no caminho de S_1 para S_2 Aqui usamos uma ligeira variação: o resultado é o melhor *mínimo local* no caminho. Um mínimo local neste contexto é uma solução que é bem-sucedida (imediatamente) e precedida (imediatamente ou por meio de um série de soluções de mesmo valor) no caminho por soluções estritamente piores. Se o caminho não tiver mínimo local, um das soluções originais (S_1 ou S_2) é retornado com igual probabilidade. Quando existe uma solução de melhoria no caminho, nosso critério corresponde exatamente ao tradicional (retornando o melhor elemento no caminho). Isto é diferente apenas quando a reconexão de caminho não tem êxito; nesse caso, tentamos aumentar a diversidade selecionando alguma solução diferente dos extremos do caminho.

Em nossa implementação, aumentamos o procedimento de intensificação realizando uma pesquisa local completa na solução produzida pela reconexão de caminho. Isso geralmente é muito mais rápido do que as aplicações em soluções construtivas aleatórias produzidas durante GRASP, uma vez que parte de uma solução que é um ótimo local ou muito próximo de ser um. Um "efeito colateral" da aplicação da pesquisa local neste ponto é o aumento da diversidade, uma vez que somos livres para usar instalações que não pertenciam a nenhuma das soluções originais.

É interessante notar que a própria reconexão de caminho é notavelmente semelhante ao procedimento de busca local descrito na Seção 4, com duas diferenças principais. Primeiro, o número de movimentos permitidos é restrito: apenas elementos em $S_2 \setminus S_1$ podem ser inseridos, e aqueles em $S_1 \setminus S_2$ pode ser removido. Em segundo lugar, movimentos que não melhoram são permitidos. No entanto, essas diferenças são sutis o suficiente para serem facilmente incorporadas na implementação básica do procedimento de pesquisa local. Em nossa implementação, ambos os procedimentos compartilham muito de seu código.

5.2 Gerenciamento de piscina

Um aspecto importante do algoritmo é o gerenciamento do pool de soluções de elite. Empiricamente, observamos que uma aplicação de reconexão de caminho a um par de soluções tem menos probabilidade de ser bem-sucedida se as soluções forem muito semelhantes. Quanto mais longo o caminho entre as soluções, maior a probabilidade de que um ótimo local totalmente diferente (em oposição às próprias soluções originais) seja encontrado. Portanto, é razoável levar em consideração não apenas a qualidade da solução, mas também a diversidade ao lidar com o conjunto de soluções de elite.

O pool deve suportar duas operações essenciais: adição de novas soluções (representadas pelo adicionar função na Figura 1) e seleção de uma solução para reconexão de caminho (o selecionar função no pseudo-código). Descrevemos cada um deles separadamente.

Atualizações. Para uma solução S com custo $v(S)$ para ser adicionado à piscina, duas condições devem ser atendidas. Primeiro, sua diferença simétrica de todas as soluções no pool cujo valor é menor que $v(S)$ deve ser pelo menos quatro (afinal, a reconexão de caminhos entre soluções que diferem em menos de quatro instalações não pode produzir soluções melhores do que as originais). Em segundo lugar, se a piscina estiver cheia, a solução deve ser pelo menos tão boa quanto a pior solução de elite (se a piscina não estiver cheia, isso obviamente não é necessário).

Se essas condições forem atendidas, a solução é inserida. Se a piscina não estiver cheia e a nova solução não estiver a uma distância de quatro de qualquer outra solução elite (incluindo as piores), ela é simplesmente adicionada. Caso contrário, substitui a solução mais semelhante entre aquelas de valor igual ou superior.

Seleção. Em cada iteração do algoritmo, uma solução é selecionada do pool (Figura 1, linha 6) e combinada com S, a solução encontrada mais recentemente. Uma abordagem que tem sido aplicada a outros problemas com algum grau de sucesso é selecionar uma solução uniformemente ao acaso [23]. No entanto, isso geralmente significa selecionar uma solução muito semelhante a S, portanto, tornando o procedimento improvável para encontrar boas soluções novas. Portanto, em vez de escolher soluções do pool uniformemente, nós as tomamos com probabilidades proporcionais à sua diferença em relação a S. Na Seção 5.4.3, mostramos que essa estratégia compensa.

5.3 Pós-otimização

No processo de busca de uma boa solução, uma heurística GRASP produz não um, mas vários ótimos locais diferentes, cujos valores geralmente não estão muito distantes do valor da melhor solução encontrada. O *fase pós-otimização* em nosso algoritmo tenta combinar essas soluções a fim de obter outras ainda melhores. Esta fase toma como entrada o pool de soluções de elite, cuja construção foi descrita nas seções anteriores. Todas as soluções do pool são combinadas entre si por meio da reconexão de caminhos. As soluções geradas por este processo são adicionadas a um novo pool de soluções de elite (seguindo as mesmas restrições descritas na Seção 5.2), representando um novo *geração.* Esse processo se repete até que se crie uma geração em que a melhor solução não seja melhor do que a melhor encontrada nas gerações anteriores. Uma estratégia de reconexão de caminho de multi-geração semelhante foi usada com sucesso em [1, 25].

5.4 Análise Empírica

Nesta seção, analisamos empiricamente alguns detalhes relacionados à intensificação que foram mencionados nas subseções anteriores. Em primeiro lugar, na Seção 5.4.1, mostramos como a execução da reconexão de caminhos durante o primeiro estágio do algoritmo (e não apenas no estágio de pós-otimização) pode nos ajudar a encontrar boas soluções mais rapidamente. Então, na Seção 5.4.2, examinamos a questão de qual *direção* escolher ao realizar a reconexão de caminhos entre duas soluções S_1 ou S_2: a partir de S_1 para S_2, a partir de S_2 para S_1, ou ambos? Finalmente, na Seção 5.4.3, examinamos qual estratégia de seleção deve ser usada.

5.4.1 Revinculação de caminho dentro do GRASP

Nossa implementação é tal que a solução construtiva aleatória produzida em cada iteração GRASP depende apenas da semente aleatória inicial, independentemente de a reconexão de caminho ser executada ou não. Portanto, se o número de iterações for o mesmo, a adição de revinculação de caminhos não pode diminuir a qualidade da solução quando comparada a um GRASP "puro". Pode ser o caso, entretanto, que o tempo extra gasto na reconexão de caminhos, se usado para iterações GRASP padrão extras, levaria a resultados ainda melhores.

Para testar essa hipótese, pegamos algumas instâncias representativas e executamos ambas as versões da heurística (com e sem revinculação de caminho) por um período 100 vezes maior que o tempo médio necessário para executar uma iteração (construção seguida por pesquisa local) *sem* revinculação de caminhos. Pudemos, portanto, comparar as qualidades das soluções obtidas à medida que o algoritmo progredia. O algoritmo construtivo usado foi amostra. Os resultados neste teste não incluem pós-otimização. Selecionamos uma instância de cada classe (fl 1400 a partir de TSP, pmed40 a partir de ORLIB, e rw500 a partir de RW), e testado cada um com sete valores de *p,* de 10 a cerca de um terço do número de instalações (*m).* O teste foi executado 10 vezes para cada valor de *p,* com 10 sementes diferentes.

A Figura 2 se refere a uma das instâncias testadas, fl 1400 com $p = 500$. O gráfico mostra como a qualidade da solução melhora com o tempo. A qualidade e o tempo são normalizados. Os tempos são dados em múltiplos do tempo médio que leva para realizar uma iteração GRASP sem reconexão de caminho (essa média é calculada em todas as iterações de todas as 10 execuções). A qualidade da solução é dada como uma fração do valor médio da solução encontrado pela primeira iteração GRASP (sem religação de caminho). [3]

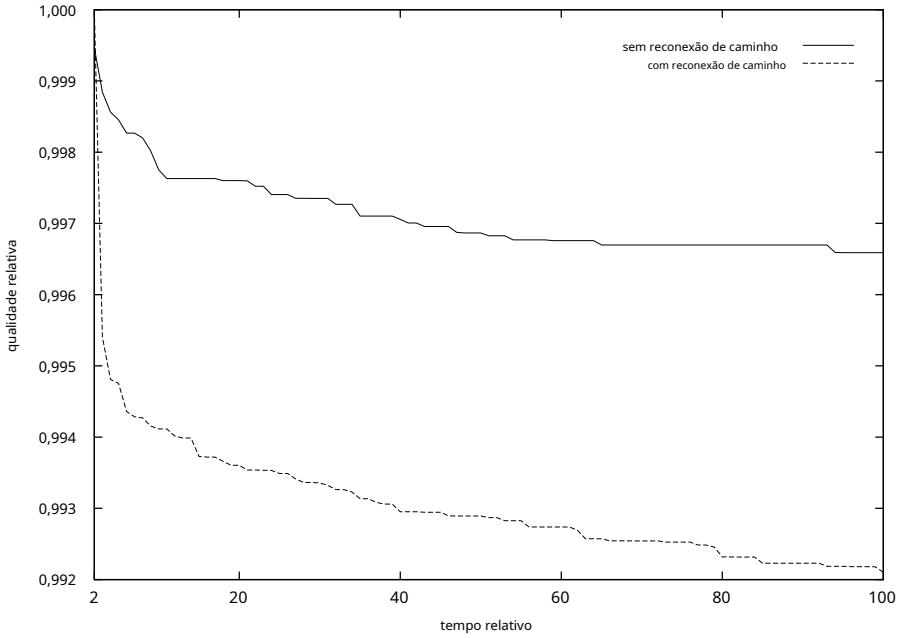

Figura 2: instância fl 1400, $p = 500$: Qualidade da melhor solução encontrada como uma fração do valor médio da primeira solução encontrada. Os tempos são dados como múltiplos do tempo médio necessário para realizar uma iteração GRASP. Valores menores são melhores.

As Figuras 3, 4 e 5 referem-se ao mesmo experimento. Cada curva nesses gráficos representa uma instância com um valor particular de *p.* Os tempos são normalizados como antes, e *relação de qualidade (* mostrado no eixo vertical) é simplesmente

[3] Observe que o primeiro valor mostrado é 2; No momento 1, nem todas as proporções são definidas, pois, em alguns casos, a primeira iteração leva mais do que o tempo médio para ser executada.

a relação entre a qualidade média da solução obtida com o path-relinking e a obtida sem ele, ambos considerando os mesmos tempos de execução. Valores menores que 1.000 favorecem a reconexão de caminhos.

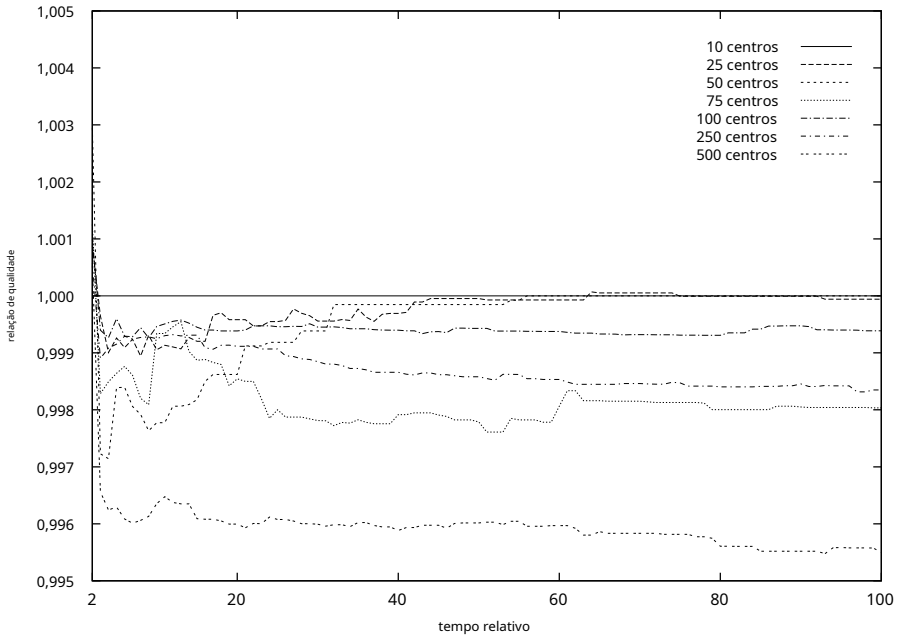

Figura 3: instância fl 1400 (aula TSP): Razões entre soluções parciais encontradas com e sem reconexão de caminho. Os tempos são normalizados em relação ao tempo médio que leva para executar uma iteração GRASP. Valores menores que 1,000 sugerem que vale a pena usar a reconexão de caminhos.

Esses resultados confirmam o que se deve esperar. Se apenas algumas iterações forem executadas, a reconexão de caminhos não é especialmente útil; soluções de qualidade comparável (ou até melhor) podem ser encontradas usando GRASP sozinho. No entanto, se for necessário gastar mais tempo, usar a reconexão de caminhos é uma boa estratégia, levando de forma consistente a soluções de qualidade superior dentro do mesmo prazo. Isso é especialmente verdadeiro para casos mais difíceis, aqueles em que p tem um valor relativamente alto. O caso da instância rw500 parece ser uma exceção; como p torna-se maior que 75, a instância aparentemente se torna mais fácil.

5.4.2 Direção

Um aspecto importante da reconexão de caminhos é o *direção* em que é executado. Dadas duas soluções S_1 e S_2, devemos decidir em qual direção realizar o relink: de S_1 para S_2, a partir de S_2 para S_1, ou ambos. Testamos os seguintes critérios:

- aleatória: Direção escolhida uniformemente ao acaso.
- pra cima: Da melhor para a pior solução (entre as duas); isso tem a vantagem potencial de explorar com mais cuidado a vizinhança mais promissora.
- baixa: Da pior para a melhor solução; explorando mais cuidadosamente a vizinhança da pior solução, pode ser capaz de encontrar boas soluções que estão relativamente longe das soluções mais conhecidas, favorecendo assim a diversidade.
- novo: Sempre comece com a solução recém-gerada, não com aquela que já está no pool (esta estratégia se aplica apenas ao primeiro estágio do algoritmo, não à fase de pós-otimização). Novamente, o objetivo é obter maior diversidade de soluções.

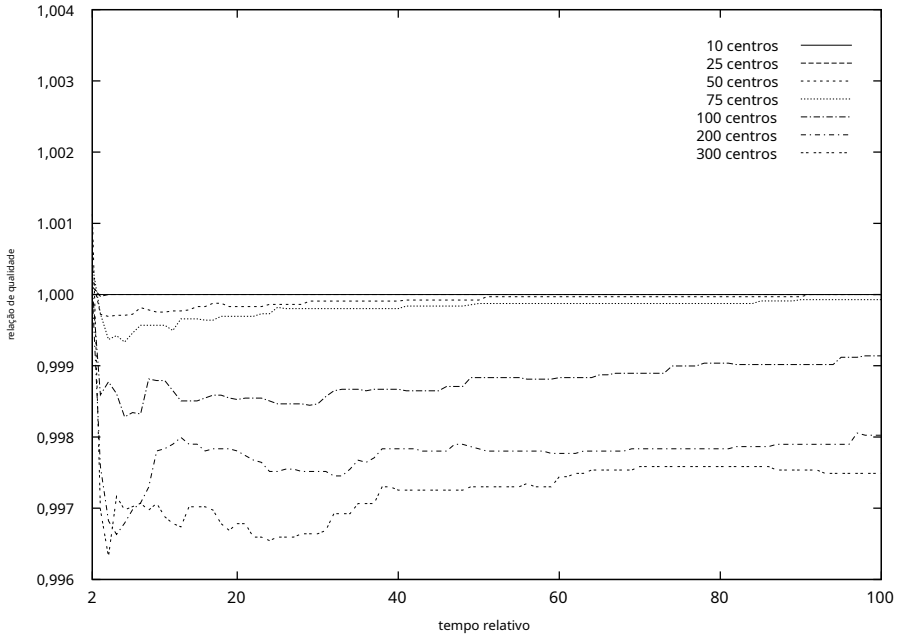

Figura 4: instância pmed40 (aula ORLIB): Razões entre soluções parciais encontradas com e sem reconexão de caminho. Os tempos são normalizados em relação ao tempo médio que leva para executar uma iteração GRASP. Valores menores que 1,000 sugerem que vale a pena usar a reconexão de caminhos.

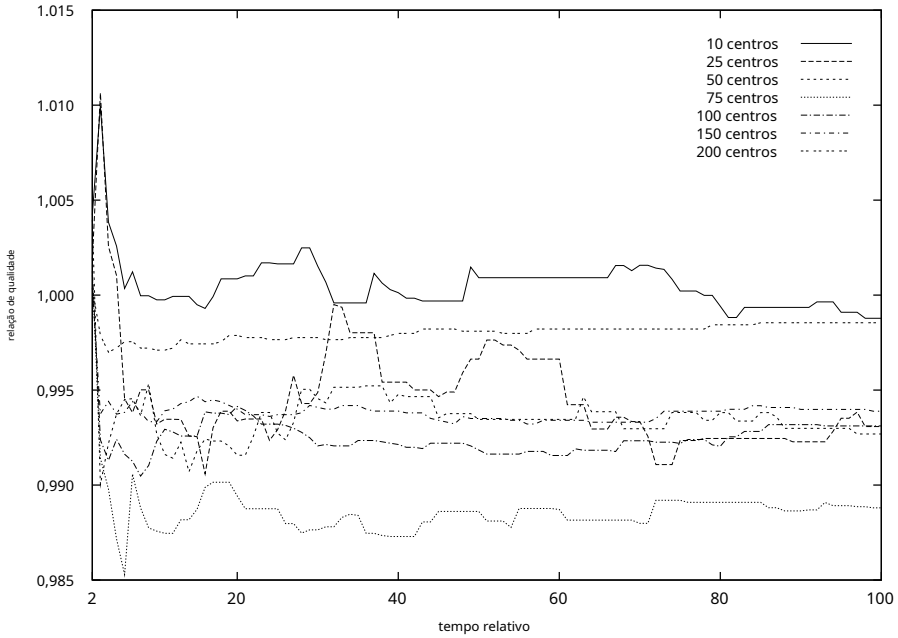

Figura 5: instância rw500 (aula RW): Razões entre soluções parciais encontradas com e sem reconexão de caminho. Os tempos são normalizados em relação ao tempo médio que leva para executar uma iteração GRASP. Valores menores que 1,000 sugerem que vale a pena usar a reconexão de caminhos.

- Nenhum: Não execute revinculação de caminho (não se aplica à pós-otimização).
- Ambas: Realize a reconexão de caminhos em ambas as direções e retorne o melhor resultado. Este método encontra os melhores resultados possíveis em cada caso, mas leva o dobro do tempo que os outros métodos executam.

Testamos todas as combinações válidas desses métodos nas 10 instâncias do conjunto restrito definido na Seção 2.1, cada uma com 10 sementes aleatórias diferentes. Em cada caso, rodamos GRASP com 32 iterações e 10 soluções elite, usando amostra como o método construtivo. As tabelas 2, 3 e 4 mostram os resultados obtidos no experimento. (As definições de *desvio percentual médio relativo* e *classificação relativa normalizada,* usados nessas tabelas, são fornecidos na Seção 3.)

Tabela 2: Qualidade da solução GRASP com diferentes estratégias de reconexão de caminho: Desvios percentuais relativos médios. Cada valor representa o quanto o valor médio da solução encontrado por cada método está acima (ou abaixo) da média encontrada por todos os métodos. Valores menores são melhores.

APERTO	MÉTODO DE PÓS-OTIMIZAÇÃO			
MÉTODO	Ambas	baixa	aleatória	pra cima
Nenhum	0,056	0,056	0,033	0,024
Ambas	0,005	0,009	- 0,030	- 0,007
baixa	- 0,010	0,007	- 0,009	- 0,012
aleatória	0,001	0,004	- 0,002	0,001
novo	- 0,008	0,004	- 0,007	- 0,011
pra cima	- 0,029	- 0,032	- 0,019	- 0,022

Tabela 3: Qualidade da solução GRASP com diferentes estratégias de reconexão de caminho: Ranks normalizados médios. Valores menores são melhores.

APERTO	MÉTODO DE PÓS-OTIMIZAÇÃO			
MÉTODO	Ambas	baixa	aleatória	pra cima
Nenhum	- 0,017	0,565	0,448	0,465
Ambas	- 0,117	- 0,143	- 0,270	0,174
baixa	- 0,357	0,270	- 0,265	0,004
aleatória	- 0,183	0,209	- 0,100	0,161
novo	- 0,387	- 0,030	- 0,135	0,078
pra cima	- 0,283	- 0,209	- 0,061	0,183

Tabela 4: Tempos de execução GRASP com diferentes estratégias de reconexão de caminho: Desvio percentual relativo médio em relação à média.

APERTO	MÉTODO DE PÓS-OTIMIZAÇÃO			
MÉTODO	Ambas	baixa	aleatória	pra cima
Nenhum	33,3	- 12,2	- 7,3	- 6,9
Ambas	27,8	- 2,3	- 0,5	- 2,3
baixa	22,7	- 9,4	- 7,8	- 9,4
aleatória	20,1	- 12,0	- 9,7	- 10,9
novo	20,3	- 8,7	- 8,0	- 11,1
pra cima	23,1	- 9,3	- 10,0	- 9,6

Observe que algumas estratégias podem ser descartadas por serem muito lentas, sem qualquer melhoria clara na qualidade da solução; esse é o caso daqueles que usam estratégia Ambas na fase de pós-otimização (e, também durante a primeira

estágio do algoritmo, embora o tempo extra neste caso seja muito menos relevante). Além disso, parece claro que usar a reconexão de caminhos durante o primeiro estágio é importante; mesmo que ainda seja possível obter soluções acima da média, eventualmente, se Nenhum é usado na primeira fase, isso só pode acontecer se Ambas é a estratégia usada na fase pós-pós-otimização - o que resulta em tempos de execução muito mais longos.

Quando as outras estratégias são consideradas, os resultados não mostram nenhuma claramente dominante. Várias combinações de novo, para cima, para baixo, e aleatória parecem escolhas razoáveis para o algoritmo. De todas as combinações que não usam o método mais demorado Ambas, cinco têm qualidade melhor do que a média de acordo com as duas medidas utilizadas: cima baixo (o primeiro método se refere ao primeiro estágio do algoritmo, o segundo ao estágio pós-otimização), baixo: aleatório, aleatório: aleatório, novo: aleatório, e up: random (consulte as Tabelas 2 e 3). Decidimos usar cima baixo na versão final do nosso algoritmo, visto que este foi o método com o melhor desvio percentual relativo médio e uma boa classificação média. Este método tem a característica interessante de favorecer a qualidade ao lidar com soluções de qualidade inferior (durante o primeiro estágio do algoritmo), e diversidade quando a qualidade geral da solução é superior (durante o estágio de pós-otimização).

5.4.3 Estratégia de Seleção

Mostramos que a aplicação de religação de caminho durante o primeiro estágio do algoritmo ajuda a encontrar boas soluções mais rapidamente. Aqui, analisamos o critério usado para escolher as soluções de elite a serem combinadas com S, a solução obtida após a pesquisa local. Lembre-se de que o método usual é selecionar a solução uniformemente ao acaso, e que propomos soluções de picking com probabilidades proporcionais à sua diferença simétrica em relação a S. Vamos chamar esses métodos uniforme e enviesado, respectivamente.

Ao realizar a reconexão de caminhos entre um par de soluções, nosso objetivo é obter uma terceira solução de menor custo. Vamos considerar a combinação *bem sucedido* quando isso acontece. O objetivo final do esquema de seleção é encontrar, entre as soluções de elite, uma que leve a uma combinação bem-sucedida. Melhores esquemas de seleção encontrarão tal solução com maior probabilidade.

Projetamos o seguinte experimento, executado para cada uma das 10 instâncias no conjunto restrito definido na Seção 2.1, para determinar qual método é melhor de acordo com esse critério. Primeiro, execute GRASP (sem reconexão de caminho) até que um pool com capacidade para 110 soluções seja preenchido. Em seguida, pegue as 10 principais soluções (ligue eles $E_1, E_2, \ldots, E_{10}$) obtido e criar um novo pool. Denote as 100 soluções restantes por $S_1, S_2, \ldots, S_{100}$ Realize a reconexão de caminhos entre cada uma dessas 100 soluções e cada solução no pool e decida com base sobre os resultados que método de seleção (enviesado ou uniforme) teria uma probabilidade maior de sucesso se tivéssemos que selecionar uma das 10 instâncias.

Para calcular a probabilidade de sucesso de cada método, precisamos de algumas definições. Deixar $s(i, j)$ seja 1 se o reconexão de caminho entre S_{eu} e E_j é bem-sucedido e 0 caso contrário; também, deixe $\Delta(eu\ j)$ ser a diferença simétrica entre S_{eu} e E_j. Para uma determinada solução S_{eu}, a probabilidade de sucesso para uniforme, se fosse aplicado, seria

$$você_i = \frac{\sum_{j=1}^{10} s(i, j)}{10}.$$

Por outro lado, a probabilidade de sucesso de enviesado seria

$$b_i = \frac{\sum_{j=1}^{10} [s(i, j) \cdot \Delta(eu\ j)]}{\sum_{i=1}^{10} \Delta(eu\ j)}.$$

Para cada uma das 10 instâncias, o procedimento descrito acima foi executado 10 vezes, com 10 sementes aleatórias, sempre utilizando amostra como o algoritmo construtivo e pra cima como a direção de reconexão do caminho. Portanto, para cada instância, 1.000 seleções foram simuladas (100 para cada semente aleatória).

Os resultados estão resumidos na Tabela 5. Para cada instância, mostramos a porcentagem de casos em que um método tem maior probabilidade de sucesso do que o outro (quando as probabilidades são iguais, consideramos o experimento um empate).

Observe que em todos os casos enviesado tem desempenho superior, às vezes por uma margem significativa. É interessante notar que em dois casos a probabilidade de empate era de quase 100%; isso se deve ao fato de que a reconexão de caminhos

Tabela 5: Comparação entre o uniforme e enviesado esquemas de seleção. Os valores representam a porcentagem de casos em que um método tem maior probabilidade de levar a uma reconexão bem-sucedida do que o outro.

INSTÂNCIA		MÉTODO DE SELEÇÃO		
NOME	*p*	uniforme	GRAVATA	enviesado
fl 1400	150	38,7	10,8	50,5
fl 1400	500	0,0	99,9	0,1
gr150	25	34,9	5,6	59,5
pcb3038	30	45,2	5,4	49,4
pcb3038	250	0,2	98,5	1,3
pmed15	100	14,5	4,5	81,0
pmed40	90	14,2	5,2	80,6
rw500	25	39,8	10,3	49,9
rw500	75	32,0	11,3	56,7
sl700	233	6,4	56,2	37,4

quase sempre funciona para essas instâncias particulares - qualquer esquema de seleção seria bem-sucedido. Nas poucas situações em que havia seleções "erradas", enviesado era melhor evitá-los.

6 resultados finais

Esta seção apresenta os resultados detalhados obtidos pela versão final de nosso método, construída com base nos experimentos relatados nas seções anteriores. Usa amostra como a heurística construtiva aleatória (consulte a Seção 3); a reconexão de caminhos é executada em ambos os estágios do algoritmo (Seção 5.4.1); A reconexão de caminhos é realizada da melhor para a pior solução durante o primeiro estágio do algoritmo, e da pior para a melhor durante a pós-otimização (Seção 5.4.2); e as soluções são selecionadas do pool de maneira tendenciosa durante o primeiro estágio do algoritmo (Seção 5.4.3). Os resultados relatados aqui se referem a execuções com 32 iterações GRASP e 10 soluções de elite - é claro, esses números podem ser alterados para tornar o algoritmo mais rápido (se forem reduzidos) ou para obter melhores soluções (se forem aumentados).

Apresentamos os resultados de todas as instâncias mencionadas na Seção 2.1. Como o nosso algoritmo é aleatório, diferentes execuções podem produzir resultados diferentes. Para evitar resultados anômalos, o executamos nove vezes para cada instância testada. Todos os valores de solução apresentados aqui referem-se ao *mediana* dos resultados obtidos (isso garante que o valor mostrado é de fato o valor de uma solução válida para o problema), enquanto os tempos são meios aritméticos assumidos sobre todas as execuções. Os resultados são apresentados nas Tabelas 6 a 12.

Para referência, as tabelas contêm os limites superiores mais baixos dos valores de solução apresentados na literatura (até onde sabemos) para cada uma das instâncias testadas. Os valores ideais são conhecidos para todas as instâncias em três classes: ORLIB (fornecido em [2]), SL [32], e GR [32]. Para classe TSP, apresentamos os melhores limites superiores listados na literatura, bem como referências aos artigos que primeiro apresentaram os limites mostrados (eles são apresentados na FONTE coluna nas Tabelas 6, 7 e 8). Embora esse seja frequentemente o caso, os limites não correspondem necessariamente às soluções encontradas pelas principais heurísticas descritas nesses documentos - em alguns casos, eles foram encontrados por outros métodos mais demorados. Para vários casos, pelo menos uma das nove execuções de nosso procedimento GRASP foi capaz de melhorar o melhor limite conhecido. Quando for esse o caso, o limite melhorado é apresentado e a coluna de origem é marcada por um traço (-). Esses valores não devem ser considerados o “resultado final” do nosso método quando comparados a outros, pois representam execuções especialmente bem-sucedidas; os resultados significativos são os *medianas* listados nas tabelas. Porque classe RW foi introduzido recentemente [24], não há resultados obtidos por heurísticas na literatura. Portanto, o MELHOR A coluna da Tabela 12 apresenta as melhores soluções encontradas por nosso algoritmo quando todas as nove execuções são consideradas.

Observe que nosso método encontrou soluções dentro de 0,2% das soluções anteriores mais conhecidas em todos os casos. Houve maiores desvios por classe RW, mas observe que eles são calculados em relação às soluções encontradas pelo próprio GRASP. No entanto, isso sugere que nosso método é capaz de obter melhores resultados - em

Tabela 6: Resultados finais para fl 1400, uma instância euclidiana da classe TSP com 1400 nós.

	MAIS CONHECIDO		APERTO DE UM ESTÁGIO			COM PÓS-OTIMIZAÇÃO		
p	VALOR	FONTE	VALOR	% ERRAR	TEMPO	VALOR	% ERRAR	TEMPO
10	101249,47	[15]	101249,55	0,000	117,1	101249,55	0,000	118,5
20	57857,55	[15]	57857,94	0,001	76,8	57857,94	0,001	83,5
30	44013,48	-	44013,48	0,000	76,0	44013,48	0,000	106,2
40	35002,52	-	35002,60	0,000	68,6	35002,60	0,000	101,3
50	29089,78	[15]	29090,23	0,002	58,8	29090,23	0,002	73,9
60	25161,12	-	25166,91	0,023	57,0	25164,02	0,012	91,5
70	22125,53	[15]	22126,03	0,002	50,6	22126,03	0,002	70,2
80	19872,72	-	19878,45	0,029	49,8	19876,57	0,019	78,1
90	17987,94	[15]	18006,83	0,105	48,7	17988,60	0,004	74,2
100	16551,20	[14]	16567,01	0,096	47,3	16559,82	0,052	82,4
150	12026,47	-	12059,12	0,271	48,7	12036,00	0,079	132,5
200	9359,15	-	9367,98	0,094	49,4	9360,67	0,016	101,3
250	7741,51	-	7754,50	0,168	54,5	7746,31	0,062	130,3
300	6620,92	-	6637,81	0,255	57,8	6623,98	0,046	167,1
350	5720,91	-	5749,51	0,500	59,6	5727,17	0,109	177,6
400	5006,83	-	5033,96	0,542	64,1	5010,22	0,068	157,5
450	4474,96	-	4485,16	0,228	68,3	4476,68	0,038	170,7
500	4047,90	-	4059,16	0,278	71,9	4049,56	0,041	210,9

Tabela 7: Resultados finais para pcb3038, uma instância euclidiana da classe TSP com 3038 nós.

	MAIS CONHECIDO		APERTO DE UM ESTÁGIO			COM PÓS-OTIMIZAÇÃO		
p	VALOR	FONTE	VALOR	% ERRAR	TEMPO	VALOR	% ERRAR	TEMPO
10	1213082,03	-	1213082,03	0,000	1115,8	1213082,03	0,000	1806,3
20	840844,53	-	840844,53	0,000	647,9	840844,53	0,000	943,4
30	677306,76	-	678108,52	0,118	426,7	677436,66	0,019	847,0
40	571887,75	-	572012,44	0,022	312,6	571887,75	0,000	492,6
50	507582,13	-	507754,72	0,034	251,7	507663,80	0,016	472,4
60	460771,87	-	461194,61	0,092	218,2	460797,55	0,006	481,4
70	426068,24	-	426933,75	0,203	201,3	426153,31	0,020	470,9
80	397529,25	-	398405,57	0,220	188,5	397585,89	0,014	555,9
90	373248,08	-	374152,75	0,242	182,3	373488,82	0,064	380,8
100	352628,35	-	353576,86	0,269	174,0	352755,13	0,036	448,1
150	281193,96	[34]	282044,70	0,303	163,3	281316,82	0,044	402,5
200	238373,26	-	238984,42	0,256	162,0	238428,35	0,023	406,9
250	209241,25	[34]	209699,36	0,219	171,8	209326,83	0,041	407,5
300	187712,12	-	188168,32	0,243	184,4	187763,64	0,027	395,8
350	170973,34	[34]	171443,87	0,275	200,0	171048,03	0,044	412,0
400	157030,46	[34]	157414,79	0,245	203,4	157073.20	0,027	436,3
450	145384,18	-	145694,26	0,213	216,3	145419,81	0,025	462,3
500	135467,85	[34]	135797,08	0,243	231,1	135507,73	0,029	478,5
550	126863,30	-	127207,83	0,272	243,8	126889,89	0,021	514,0
600	119107,99	[15]	119428,60	0,269	258,3	119135,62	0,023	595,8
650	112063,73	-	112456,15	0,350	271,0	112074,74	0,010	619,0
700	105854,40	-	106248,00	0,372	284,0	105889,22	0,033	637,3
750	100362,55	[15]	100713,79	0,350	296,4	100391,53	0,029	649,3
800	95411,78	-	95723,00	0,326	286,6	95432,66	0,022	677,8
850	91003,62	-	91268,56	0,291	296,1	91033,10	0,032	689,3
900	86984,10	-	87259,78	0,317	306,4	87022,59	0,044	730,4
950	83278,78	-	83509,58	0,277	314,3	83299,22	0,025	780,5
1000	79858,79	-	80018,33	0,200	321,7	79869,98	0,014	806,2

Tabela 8: Resultados finais, por exemplo rl5934, uma instância euclidiana da classe TSP com 5934 nós.

	MAIS CONHECIDO		APERTO DE UM ESTÁGIO			COM PÓS-OTIMIZAÇÃO		
p	VALOR	FONTE	VALOR	% ERRAR	TEMPO	VALOR	% ERRAR	TEMPO
10	9794951,00	[15]	9794973,65	0,000	5971,1	9794973,65	0,000	8687,1
20	6718848,19	-	6719116,39	0,004	3296,8	6719026,03	0,003	4779,6
30	5374936,14	-	5379979,09	0,094	2049,8	5376040,45	0,021	4515,1
40	4550364,60	-	4550843,75	0,011	1470,4	4550518,95	0,003	2499,3
50	4032379,97	-	4033758,13	0,034	1195,3	4032675,94	0,007	2280,6
60	3642397,88	-	3646198,03	0,104	996,1	3642949,30	0,015	2244,0
70	3343712,45	-	3348834,92	0,153	872,5	3344888,24	0,035	2138,3
80	3094824,49	-	3099917,93	0,165	778,8	3095442.55	0,020	1792,4
90	2893362,39	-	2898721,66	0,185	708,8	2894954,78	0,055	1844,2
100	2725180,81	-	2730313,90	0,188	671,2	2725580,72	0,015	1892,6
150	2147881,53	-	2151985,53	0,191	560,2	2148749,47	0,040	1209,2
200	1808179,07	-	1812249,63	0,225	526,6	1808658,73	0,027	1253,0
250	1569941,34	-	1573800,83	0,246	526,2	1570445,77	0,032	1203,8
300	1394115,39	-	1397064,23	0,212	550,1	1394361,41	0,018	1042,7
350	1256844,04	-	1259733,85	0,230	575,6	1257098,17	0,020	1246,4
400	1145669,38	[15]	1148386,49	0,237	583,8	1145961,13	0,025	1157,6
450	1053363,64	-	1055756,67	0,227	619,2	1053729,79	0,035	1236,9
500	973995,18	-	975940,78	0,200	641,7	974242.08	0,025	1236,7
600	848283,85	-	849765,46	0,175	703,7	848499,21	0,025	1439,4
700	752068,38	[15]	753522,21	0,193	767,3	752263,82	0,026	1566,6
800	676795,78	-	678300,99	0,222	782,1	676956,64	0,024	1574,9
900	613367,44	[15]	614506,49	0,186	834,5	613498,64	0,021	1722,0
1000	558802,38	[15]	559797,83	0,178	877,7	558943,93	0,025	1705,3
1100	511813,19	[15]	512793.56	0,192	931,4	511928,86	0,023	1893,4
1200	470295,38	[15]	471486,76	0,253	988,1	470411,12	0,025	2082,0
1300	433597,44	[15]	434688,75	0,252	1033,4	433678,02	0,019	2147,8
1400	401853,00	[15]	402796,80	0,235	1072,4	401934,24	0,020	2288,7
1500	374014,57	-	374803,24	0,211	1029,7	374056,40	0,011	2230,3

Tabela 9: Resultados finais obtidos para a aula ORLIB, instâncias baseadas em grafos introduzidas por Beasley [2].

INSTÂNCIA				APERTO DE UM ESTÁGIO			COM PÓS-OTIMIZAÇÃO		
NOME	*n*	*p*	OPTAR	VALOR	% ERRAR	TEMPO	VALOR	% ERRAR	TEMPO
pmed01	100	5	5819	5819	0,000	0,5	5819	0,000	0,5
pmed02	100	10	4093	4093	0,000	0,4	4093	0,000	0,5
pmed03	100	10	4250	4250	0,000	0,4	4250	0,000	0,5
pmed04	100	20	3034	3034	0,000	0,4	3034	0,000	0,5
pmed05	100	33	1355	1355	0,000	0,4	1355	0,000	0,5
pmed06	200	5	7824	7824	0,000	1,8	7824	0,000	1,8
pmed07	200	10	5631	5631	0,000	1,4	5631	0,000	1,4
pmed08	200	20	4445	4445	0,000	1,2	4445	0,000	1,2
pmed09	200	40	2734	2734	0,000	1,2	2734	0,000	1,5
pmed10	200	67	1255	1255	0,000	1,3	1255	0,000	1,6
pmed11	300	5	7696	7696	0,000	3,5	7696	0,000	3,5
pmed12	300	10	6634	6634	0,000	2,9	6634	0,000	2,9
pmed13	300	30	4374	4374	0,000	2,4	4374	0,000	2,5
pmed14	300	60	2968	2968	0,000	2,9	2968	0,000	3,5
pmed15	300	100	1729	1729	0,000	3,3	1729	0,000	4,3
pmed16	400	5	8162	8162	0,000	8,1	8162	0,000	8,2
pmed17	400	10	6999	6999	0,000	6,1	6999	0,000	6,3
pmed18	400	40	4809	4809	0,000	5,5	4809	0,000	6,7
pmed19	400	80	2845	2845	0,000	6,3	2845	0,000	7,5
pmed20	400	133	1789	1789	0,000	7,1	1789	0,000	8,6
pmed21	500	5	9138	9138	0,000	12,2	9138	0,000	12,2
pmed22	500	10	8579	8579	0,000	10,7	8579	0,000	11,3
pmed23	500	50	4619	4619	0,000	9,4	4619	0,000	11,0
pmed24	500	100	2961	2961	0,000	11,4	2961	0,000	13,1
pmed25	500	167	1828	1828	0,000	13,4	1828	0,000	16,2
pmed26	600	5	9917	9917	0,000	20,5	9917	0,000	20,5
pmed27	600	10	8307	8307	0,000	16,4	8307	0,000	16,4
pmed28	600	60	4498	4498	0,000	14,6	4498	0,000	17,4
pmed29	600	120	3033	3033	0,000	18,0	3033	0,000	21,0
pmed30	600	200	1989	1989	0,000	21,1	1989	0,000	26,9
pmed31	700	5	10086	10086	0,000	28,8	10086	0,000	28,8
pmed32	700	10	9297	9297	0,000	22,8	9297	0,000	22,9
pmed33	700	70	4700	4700	0,000	20,6	4700	0,000	23,7
pmed34	700	140	3013	3013	0,000	25,8	3013	0,000	30,8
pmed35	800	5	10400	10400	0,000	36,7	10400	0,000	36,7
pmed36	800	10	9934	9934	0,000	31,7	9934	0,000	34,4
pmed37	800	80	5057	5057	0,000	28,8	5057	0,000	32,4
pmed38	900	5	11060	11060	0,000	52,9	11060	0,000	52,9
pmed39	900	10	9423	9423	0,000	36,5	9423	0,000	36,5
pmed40	900	90	5128	5129	0,020	36,6	5128	0,000	43,4

Tabela 10: Resultados finais para a aula SL, instâncias baseadas em grafos introduzidas por Senne e Lorena [32].

INSTÂNCIA				APERTO DE UM ESTÁGIO			COM PÓS-OTIMIZAÇÃO		
NOME	*n*	*p*	OPTAR	VALOR	% ERRAR	TEMPO	VALOR	% ERRAR	TEMPO
sl700	700	233	1847	1848	0,054	30,2	1847	0,000	39,5
sl800	800	267	2026	2027	0,049	41,8	2026	0,000	53,2
sl900	900	300	2106	2107	0,047	54,1	2106	0,000	68,2

Tabela 11: Resultados finais para a aula GR, instâncias baseadas em grafos introduzidas por Galvão e ReVelle [9].

INSTÂNCIA			APERTO DE UM ESTÁGIO			COM PÓS-OTIMIZAÇÃO		
NOME	p	OPTAR	VALOR	% ERRAR	TEMPO	VALOR	% ERRAR	TEMPO
gr100	5	5703	5703	0,000	0,5	5703	0,000	0,5
	10	4426	4426	0,000	0,6	4426	0,000	1.0
	15	3893	3893	0,000	0,5	3893	0,000	0,8
	20	3565	3565	0,000	0,4	3565	0,000	0,7
	25	3291	3291	0,000	0,4	3291	0,000	0,7
	30	3032	3032	0,000	0,4	3032	0,000	0,6
	40	2542	2542	0,000	0,4	2542	0,000	0,6
	50	2083	2083	0,000	0,4	2083	0,000	0,6
gr150	5	10839	10839	0,000	1,3	10839	0,000	1,3
	10	8729	8729	0,000	1,1	8729	0,000	2.0
	15	7390	7390	0,000	1.0	7390	0,000	1,7
	20	6454	6462	0,124	0.9	6462	0,124	1,5
	25	5875	5887	0,204	0.9	5875	0,000	1,7
	30	5495	5502	0,127	0,8	5495	0,000	1,5
	40	4907	4907	0,000	0,8	4907	0,000	1,2
	50	4374	4375	0,023	0,8	4375	0,023	1,2

Tabela 12: Resultados finais para a aula RW, instâncias aleatórias introduzidas em [24].

INSTÂNCIA			APERTO DE UM ESTÁGIO			COM PÓS-OTIMIZAÇÃO		
NOME	p	MELHOR	VALOR	% ERRAR	TEMPO	VALOR	% ERRAR	TEMPO
rw100	10	530	530	0,000	0,7	530	0,000	1,3
	20	277	277	0,000	0,5	277	0,000	0,7
	30	213	213	0,000	0,4	213	0,000	0,5
	40	187	187	0,000	0,3	187	0,000	0,5
	50	172	172	0,000	0,3	172	0,000	0,4
rw250	10	3691	3691	0,000	6,1	3691	0,000	10,4
	25	1364	1370	0,440	3,3	1364	0,000	5,8
	50	713	718	0,701	2,1	713	0,000	3,9
	75	523	523	0,000	1,9	523	0,000	2,6
	100	444	444	0,000	1,8	444	0,000	2,2
	125	411	411	0,000	1,5	411	0,000	2.0
rw500	10	16108	16259	0,937	33,1	16108	0,000	76,9
	25	5681	5749	1,197	20,8	5683	0,035	46,9
	50	2628	2657	1,104	14,1	2635	0,266	27,7
	75	1757	1767	0,569	11,6	1757	0,000	20,5
	100	1380	1388	0,580	11,5	1382	0,145	20,4
	150	1024	1026	0,195	11,1	1024	0,000	15,4
	200	893	893	0,000	11,8	893	0,000	14,4
	250	833	833	0,000	9,6	833	0,000	11,6
rw1000	10	67811	68202	0,577	153,6	68136	0,479	256,3
	25	24896	25192	1,189	111,1	24964	0,273	293,5
	50	11306	11486	1.592	77,7	11360	0,478	169,1
	75	7161	7302	1.969	60,2	7207	0,642	160,1
	100	5223	5297	1.417	55,5	5259	0,689	109,8
	200	2706	2727	0,776	57,5	2710	0,148	100,4
	300	2018	2021	0,149	55,2	2018	0,000	71,5
	400	1734	1734	0,000	61,8	1734	0,000	73,5
	500	1614	1614	0,000	47,9	1614	0,000	55,9

termos absolutos - para instâncias com métricas bem definidas (gráficos e instâncias euclidianas), do que em instâncias aleatórias (tal classe RW). Isso não é surpreendente, apenas reflete o comportamento dos blocos de construção básicos de nosso método, o procedimento de busca local e o método de reconexão de caminho.

Outros métodos. Agora analisamos nossos métodos em termos relativos: como se comporta em comparação com outros métodos na literatura. Para simplificar, nos referimos ao nosso próprio método como GRASP, embora os resultados relatados aqui incluam a reconexão de caminhos e a fase de pós-otimização. Os resultados considerados na comparação são os apresentados nas três últimas colunas das Tabelas 6 e 12.

Outros métodos mencionados na comparação são:

- VNS: Variable Neighborhood Search, de Hansen e Mladenović´ [14]; resultados para este método estão disponíveis para o ORLIB série (todas as 40 instâncias foram testadas, com tempos de execução fornecidos para apenas 22 delas), para fl 1400 (todos os 18 valores de *p),* e pcb3038 (com apenas 10 valores de *p:* 50.100.150,. . . , 500). Os valores usados aqui foram calculados daqueles relatados nas Tabelas 1, 2 e 3 de [14].

- VNDS: Variable Neighborhood Decomposition Search, de Hansen, Mladenović´ e Perez-Brito [15]. Os resultados estão disponíveis para todos ORLIB e TSP instâncias. [4]

- LOPT: Método de otimização local, proposto por Taillard em [34]. O método funciona resolvendo heuristicamente subproblemas definidos localmente, integrando-os em uma solução para o problema geral. O autor fornece resultados detalhados (na Tabela 7 de [34]) apenas por exemplo pcb3038, com nove valores de *p,* todos os múltiplos de 50 entre 100 a 500.

- DEC: Procedimento de Decomposição, também estudado por Taillard em [34] e baseado na decomposição do problema original. Os resultados são fornecidos pelas mesmas nove instâncias do LOPT (na Tabela 7 de [34]).

- LSH: Lagrangean-Surrogate Heuristic, descrito por Senne e Lorena em [32]. O papel contém resultados para seis ORLIB instâncias (pmed05, pmed10, pmed15, pmed20, pmed25, pmed30), para nove valores de *p* para pcb3038 (os mesmos nove usados para testar LOPT), e para todas as instâncias em classes SL e GR. Os valores usados em nossa comparação foram retirados das Tabelas 1, 2 e 3 de [32].

- CGLS: ColumnGenerationwith Lagrangean / SurrogateRelaxation, estudado por Senne e Lorena em [33]. Os resultados estão disponíveis para 15 ORLIB instâncias (pmed01, pmed05, pmed06, pmed07, pmed10, pmed11, pmed12, pmed13, pmed15, pmed16, pmed17, pmed18, pmed20, pmed25, e pmed30), para todos os três SL instâncias, e para cinco valores de *p* por exemplo pcb3038 (300, 350, 400, 450 e 500). Consideramos aqui os resultados encontrados pelo método *CG (t),* retirado das Tabelas 1, 2 e 4 de [33].

Iniciamos nossa comparação apresentando, para cada um dos métodos estudados, o desvio percentual médio em relação às melhores soluções conhecidas (conforme as Tabelas 6 a 12 acima). Os resultados são apresentados na Tabela 13. Cada uma das instâncias de classe TSP é mostrado separadamente para permitir uma análise mais precisa dos algoritmos. Alguns valores estão em *fonte inclinada* como um lembrete de que nem todas as instâncias do conjunto foram consideradas no artigo que descreve o método. Um traço (-) é mostrado quando nenhum resultado para a classe está disponível. Aula RW não é mostrado, uma vez que os únicos resultados disponíveis são aqueles obtidos pelo nosso método.

A tabela mostra que nosso método é o único dentro de 0,04% dos melhores valores conhecidos para todas as instâncias. Além disso, embora nosso método não tenha obtido os melhores resultados para todas as instâncias, ele obteve os melhores resultados em média para todos os seis conjuntos de instâncias apresentados na tabela. Deve-se dizer, entretanto, que a diferença costuma ser muito pequena. Muitos dos métodos são virtualmente tão bons quanto os nossos em uma ou outra classe: esse é o caso do VNDS para todos os três TSP instâncias; de VNS e LSH para ORLIB instâncias; e de CGLS para pcb3038. Isso revela a maior força do nosso método: *robustez.* Ele foi capaz de obter resultados competitivos para todas as classes de instâncias. Nenhum dos métodos testados apresentou tal grau de consistência.

Claro, também devemos considerar os tempos de execução dos métodos envolvidos. Encontrar boas soluções dificilmente seria surpreendente se os tempos de execução de nosso método fossem significativamente maiores do que os dos outros.

[4] Os autores de [15] também testaram instâncias de [26]; infelizmente, não foi possível obter essas instâncias no momento da redação.

Tabela 13: Desvios percentuais médios para cada método em relação à melhor solução conhecida. Valores em *fonte inclinada* indicam que nem todas as instâncias do conjunto foram testadas pelo método. Valores menores são melhores.

Series	APERTO	CGLS	DEZ	LOPT	LSH	VNDS	VNS
GR	0,009	-	-	-	0,727	-	-
SL	0,000	0,691	-	-	0,332	-	-
ORLIB	0,000	*0,101*	-	-	*0,000*	0,116	0,007
fl 1400	0,031	-	-	-	-	0,071	0,191
pcb3038	0,025	*0,043*	*4.120*	*0,712*	*2,316*	0,117	*0,354*
rl5934	0,022	-	-	-	-	0,142	-

Para apresentar uma comparação significativa entre os métodos, adotamos a seguinte estratégia. Para cada instância em que um método foi testado, calculamos a razão entre o tempo necessário e o tempo de execução do nosso método. Na Tabela 14, apresentamos o *meios geométricos* dessas proporções assumidas sobre as instâncias em cada conjunto (mais uma vez, apenas as instâncias testadas pelo método relevante são consideradas). Acreditamos que isso faz mais sentido do que a média aritmética usual neste caso: se um método é duas vezes mais rápido que outro para 50% das instâncias e metade mais rápido para os outros 50%, intuitivamente os métodos devem ser considerados equivalentes. A média geométrica reflete isso, enquanto a média aritmética não.

Tabela 14: Razões médias entre os tempos de corrida obtidos pelos métodos da literatura e os obtidos pelo nosso. Embora valores maiores que 1,0 indiquem que nosso método é mais rápido em média, diferenças dentro da mesma ordem de magnitude devem ser desconsideradas. Valores em *fonte inclinada* indicam que há instâncias no conjunto para as quais os horários não estão disponíveis.

Series	APERTO	CGLS	DEZ	LOPT	LSH	VNDS	VNS
GR	1,00	-	-	-	1,11	-	-
SL	1,00	0,51	-	-	24,20	-	-
ORLIB	1,00	*55,98*	-	-	*4,13*	0,46	*5,47*
fl 1400	1,00	-	-	-	-	0,58	19,01
pcb3038	1,00	*9,55*	*0,21*	*0,35*	*1,67*	2,60	*30,94*
rl5934	1,00	-	-	-	-	2,93	-

Uma observação importante quanto aos valores apresentados deve ser feita: para VNS e VNDS, os tempos considerados são momentos em que foi encontrada a melhor solução (como nos artigos que descrevem esses métodos [14, 15]); para todos os outros algoritmos (incluindo o nosso), o *total* o tempo de execução é considerado. Os valores relatados para o nosso algoritmo também incluem o tempo necessário para pré-calcular todas as distâncias dos vértices dos pares em classes baseadas em gráficos (ORLIB e SL). [5]

Valores maiores que um na tabela favorecem nosso método, enquanto valores menores que um favorecem outros. Não se deve tomar esses resultados muito literalmente, no entanto. Uma vez que os resultados foram obtidos em máquinas diferentes (ver Tabela 15), pequenas diferenças no tempo de execução não devem ser usadas para tirar qualquer conclusão sobre a eficácia relativa dos algoritmos. Os tempos de execução dentro da mesma ordem de magnitude devem ser considerados indistinguíveis.

7 Observações Finais

Neste artigo, apresentamos um GRASP com redirecionamento de caminho para o *p*- problema mediano. Mostramos que é extremamente robusto, lidando com uma grande variedade de instâncias e obtendo resultados competitivos com aqueles

[5] GR também é uma classe baseada em grafos, mas as instâncias que obtivemos, gentilmente cedidas por E. Senne, já estavam representadas como matrizes de distância.

Tabela 15: Máquinas nas quais os tempos usados na Tabela 14 foram obtidos.

MÉTODO	MÁQUINA
CGLS	Sun Ultra 30
DEZ	Estação de trabalho SGI (195 MHz MIPS R10000)
APERTO	SGI Challenge (196 MHz MIPS R10000) Estação de
LOPT	trabalho SGI (195 MHz MIPS R10000) Sun Ultra 30
LS	
VNDS	Sun Ultra I (143 MHz UltraSparc)
VNS	Sun SparcStation 10

obtidos pelas melhores heurísticas da literatura em cada caso. Isso torna nossa heurística uma candidata valiosa para um solucionador de propósito geral para o *p*- problema mediano.

Não afirmamos, entretanto, que nosso método seja o melhor em todas as circunstâncias. Outros métodos descritos na literatura são capazes de produzir resultados de qualidade notavelmente boa, muitas vezes às custas de tempos de execução um pouco maiores. VNS [14] é especialmente bem-sucedido para instâncias de gráfico; O VNDS [15] é particularmente forte para instâncias euclidianas e geralmente é mais rápido que nosso método (especialmente quando o número de instalações a serem abertas é muito pequeno); e CGLS [33] pode obter resultados muito bons para instâncias euclidianas, e tem a vantagem adicional de fornecer bons limites inferiores. Um tópico de pesquisa interessante seria usar algumas das idéias exploradas neste artigo (como uma implementação rápida do procedimento de busca local e a combinação de soluções de elite por meio de reconexão de caminhos) com esses métodos para obter resultados ainda melhores.

O objetivo do nosso algoritmo é produzir soluções quase ótimas. Portanto, deve ser dito que nosso método não lida bem com instâncias realmente grandes. Se a entrada for um gráfico com milhões de vértices, simplesmente computar os caminhos mais curtos com todos os pares seria proibitivamente lento. Para tanto, provavelmente seria melhor confiar em métodos baseados em técnicas de amostragem como a proposta por Thorup [36]. Métodos como este visam encontrar soluções que são “boas”, não quase ótimas, em um período de tempo razoável (quase linear).

No entanto, se alguém estiver interessado em resolver instâncias grandes o suficiente para impedir a aplicação de algoritmos exatos, mas não tão grandes a ponto de tornar algo pior do que uma proibição quase linear, nosso método provou ser uma alternativa muito útil.

Referências

[1] RM Aiex, MGC Resende, PM Pardalos e G. Toraldo. GRASP com revinculação de caminho para o problema de atribuição de três índices. Relatório Técnico TD-4RR83X, AT&T Labs Research, 2000.

[2] JE Beasley. Uma nota sobre a resolução de grandes *p*- problemas medianos. *European Journal of Operational Research,* 21: 270–273,1985.

[3] G. Cornuejols, ML Fisher e GL Nemhauser. Localização de contas bancárias para otimizar o fl utuador: Um estudo analítico de algoritmos exatos e aproximados. *Ciência da Administração,* 23: 789–810,1977.

[4] O. du Merle, D. Villeneuve, J. Desrosiers e P. Hansen. Geração de coluna estabilizada. *Discreto Matemática,* 194: 229–237, 1999.

[5] TA Feo e MGC Resende. Uma heurística probabilística para um conjunto computacionalmente difícil cobrindo problema. *Cartas de pesquisa operacional,* 8: 67–71,1989.

[6] TA Feo e MGC Resende. Procedimentos de busca adaptativa aleatória gananciosa. *Journal of Global Otimização,* 6: 109–133, 1995.

[7] P. Festa e MGC Resende. GRASP: Uma bibliografia comentada. Em CC Ribeiro e P. Hansen, editores, *Ensaios e pesquisas sobre metaheurísticas,* páginas 325–367. Kluwer, 2002.

[8] RD Galvão. Um algoritmo de limite duplo para o *p*- problema mediano. *Pesquisa Operacional,* 28: 1112-1121, 1980.

[9] RD Galvão e CS ReVelle. Uma heurística Lagrangeana para o problema de cobertura máxima. *europeu Journal of Operational Research,* 18: 114-123,1996.

[10] F. García-López, B. Melián-Batista, JA Moreno-Pérez e JM Moreno-Vega. A variável paralela busca na vizinhança pelo *p*- problema mediano. *Journal of Heuristics,* 8 (3): 375–388, 2002.

[11] F. Glover. Pesquisa tabu e programação de memória adaptativa: avanços, aplicações e desafios. Dentro RS Barr, RV Helgason e JL Kennington, editores, *Interfaces em Computer Scinece e Pesquisa Operacional,* páginas 1–75. Kluwer, 1996.

[12] F. Glover, M. Laguna e R. Martí. Fundamentos da pesquisa dispersa e reconexão de caminhos. *Controle e Cibernética,* 39: 653–684, 2000.

[13] MF Goodchild e V. Noronha. Alocação de localização para pequenos computadores. Monografia 8, Departamento of Geography, University of Iowa, 1983.

[14] P. Hansen e N. Mladenović´. Pesquisa de vizinhança variável para o *p*- mediana. *Ciência da localização,* 5: 207–226, 1997.

[15] P. Hansen, N. Mladenović´ e D. Perez-Brito. Pesquisa de decomposição de vizinhança variável. *Diário de heurísticas,* 7 (3): 335-350, 2001.

[16] MJ Hodgson. Rumo a uma alocação mais realista em modelos de alocação de localização: uma abordagem de interação. *Meio Ambiente e Planejamento A,* 10: 1273–85, 1978.

[17] AA Kuehn e MJ Hamburger. Um programa heurístico para localização de depósitos. *Ciência da Administração,* 9 (4): 643–666, 1963.

[18] M. Laguna e R. Martí. GRASP e reconexão de caminho para minimização de cruzamento de linha reta de 2 camadas. *INFORMS Journal on Computing,* 11: 44–52,1999.

[19] FE Maranzana. Sobre a localização dos pontos de abastecimento para minimizar os custos de transporte. *Pesquisa Operacional Trimestral,* 15 (3): 261–270, 1964.

[20] M. Matsumoto e T. Nishimura. Mersenne Twister: Um uniforme 623 dimensionalmente equidistribuído gerador de números pseudoaleatórios. *Transações ACM em modelagem e simulação de computador,* 8 (1): 3–30, 1998.

[21] RM Nauss e RE Markland. Teoria e aplicação de um procedimento de otimização para lock box análise de localização. *Ciência da Administração,* 27: 855–865,1981.

[22] G. Reinelt. TSPLIB: Uma biblioteca de problemas do caixeiro viajante. *ORSA Journal on Computing,* 3: 376-384, 1991. *http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/.*

[23] MGC Resende e CC Ribeiro. Procedimentos de busca adaptativos gananciosos e randomizados. Em F. Glover e G. Kochenberger, editores, *State of the Art Handbook in Metaheuristics.* Kluwer, 2002. Para aparecer.

[24] MGC Resende e R. Werneck. Sobre a implementação de um procedimento de pesquisa local baseado em troca para a *p*- problema mediano. Relatório técnico TD-5E4QKA, AT&T Labs Research, 2002.

[25] CC Ribeiro, E. Uchoa e RF Werneck. Um GRASP híbrido com perturbações para o Steiner problema em gráficos. *INFORMS Journal on Computing,* 14 (3): 228–246, 2002.

[26] E. Rolland, DA Schilling e JR Current. Um procedimento eficiente de pesquisa tabu para o *p*- mediana problema. *European Journal of Operational Research,* 96: 329–342, 1996.

[27] KE Rosing. Uma investigação empírica da eficácia de uma heurística de substituição de vértices. *Environ- planejamento e planejamento B,* 24: 59–67, 1997.

[28] KE Rosing e CS ReVelle. Concentração heurística: construção de solução em dois estágios. *europeu Journal of Operational Research,* 97: 75–86, 1997.

[29] KE Rosing, CS ReVelle, E. Rolland, DA Schilling e JR Current. Concentração heurística e pesquisa tabu: uma comparação direta. *European Journal of Operational Research,* 104: 93–99, 1998.

[30] KE Rosing, CS ReVelle e H. Rosing-Vogelaar. O *p*- mediana e sua programação linear relaxação: uma abordagem para grandes problemas. *Jornal da Sociedade de Pesquisa Operacional,* 30 (9): 815-823, 1979.

[31] E. Senne, 2002. Comunicação pessoal.

[32] ELF Senne e LAN Lorena. Heurísticas Langrangeanas / surrogate para *p*- problemas medianos. InM. Laguna e JL Gonzalez-Velard, editores, *Ferramentas de computação para otimização e simulação de modelagem: interfaces em ciência da computação e pesquisa operacional,* páginas 115-130. Kluwer, 2000.

[33] ELF Senne e LAN Lorena. Estabilizando a geração de colunas usando Lagrangean / surrogate relaxação: um aplicativo para *p*- problemas de localização mediana. *European Journal of Operational Research,* 2002. Para aparecer.

[34] ED Taillard. Métodos heurísticos para grandes problemas de agrupamento de centróides. Relatório Técnico 96-96, IDSIA, Lugano, Suíça, 1998. Revisado de uma versão original publicada em 1996.

[35] MB Teitz e P. Bart. Métodos heurísticos para estimar a mediana generalizada do vértice de um gráfico. *Pesquisa Operacional,* 16 (5): 955–961,1968.

[36] M. Thorup. Rápido *k*- mediana, *k*- centro e localização da instalação para gráficos esparsos. Dentro *Atas do dia 28 Colóquio Internacional sobre Autômatos, Linguagens e Programação (ICALP 2001),* volume 2076 de *Notas de aula em ciência da computação,* páginas 249–260. Springer, 2001.

[37] S. Voss. Uma abordagem de eliminação reversa para o *p*- problema mediano. *Estudos em Análise Locacional,* 8: 49–58, 1996.

[38] R. Whitaker. Um algoritmo rápido para o intercâmbio ganancioso de agrupamento em grande escala e localização mediana prolems. *INFOR,* 21: 95–108,1983.