ELSEVIER

Available online at www.sciencedirect.com
SCIENCE DIRECT®

Computers & Operations Research 31 (2004) 1985 - 2002

computers & operations research

www.elsevier.com/locate/dsw

Um algoritmo evolutivo simples e eficaz para o problema de roteamento de veículos

Christian Prins*

LOSI, Universidade de Tecnologia de Troyes, BP 2060, 10010 Troyes Cedex, França

Resumo

O problema de roteamento de veículos (VRP) desempenha um papel central na otimização de redes de distribuição. Como algumas instâncias clássicas com 75 nós resistem aos melhores métodos de solução exata, a maioria dos pesquisadores se concentra em metaheurísticas para resolver problemas reais. Ao contrário do VRP com janelas de tempo, nenhum algoritmo genético (GA) pode competir com os poderosos métodos de busca tabu (TS) projetados para o VRP. Este artigo preenche essa lacuna ao apresentar um GA híbrido relativamente simples, mas eficaz. Em termos de custo médio de solução, esse algoritmo supera a maioria das heurísticas de TS publicadas nas 14 instâncias clássicas de Christofides e se torna o melhor método de solução para as 20 instâncias de grande escala geradas por Golden et al.

Escopo e objetivo

A estrutura desta pesquisa é o desenvolvimento de metaheurísticas eficazes para problemas difíceis de otimização combinatória encontrados no roteamento de veículos. É surpreendente notar na literatura a ausência de algoritmos genéticos (AG) eficazes para o problema de roteamento de veículos (VRP, o principal problema de roteamento de nós capacitados), ao contrário dos problemas de roteamento de nós com janelas de tempo ou problemas de roteamento de arcos. As tentativas anteriores baseavam-se em cromossomos com delimitadores de viagem e precisavam de um procedimento de reparo para obter filhos viáveis após cada cruzamento. Sabe-se que esses procedimentos enfraquecem a transmissão genética de informações de pais para filhos. Este artigo propõe um AG sem delimitadores de viagem, hibridizado com um procedimento de busca local. A qualquer momento, um cromossomo pode ser convertido em uma solução VRP ideal (sujeito à sequência de cromossomos), graças a um procedimento especial de divisão. Essa escolha de projeto evita procedimentos de reparo e permite o uso de cruzamentos clássicos como o OX. O algoritmo resultante é flexível, relativamente simples e muito eficaz quando aplicado a dois conjuntos de instâncias de referência padrão que variam de 50 a 483 clientes.

Palavras-chave: Problema de roteamento de veículos; Algoritmo genético

* Tel: +33-325-71-56-41; fax: +33-325-71-56-49.
Endereço de e-mail: prins@utt.fr (C. Prins).

0305-0548/$ - see front matter

 doi:10.1016/S0305-0548(03)00158-8

1. Introdução

O problema de roteamento de veículos (VRP) é definido em uma rede completa não direcionada $G = (V, E)$ com um conjunto de nós $V = \{0, 1, \ldots, n\}$ e um conjunto de arestas E. O nó 0 é um depósito com m veículos idênticos de capacidade W, m pode ser fixado a priori ou deixado como uma variável de decisão. Cada outro nó $i > 0$ representa um cliente com uma demanda não negativa q_i e cada borda (i, j) tem um custo de viagem não negativo $c_{ij} = c_{ji}$. O VRP consiste em determinar um conjunto de m viagens de veículos com custo total mínimo, de modo que cada viagem comece e termine no depósito, cada cliente seja visitado exatamente uma vez e a demanda total atendida por qualquer veículo não exceda W. Na prática, os clientes são distribuídos em uma grande rede de estradas e o custo em cada borda é geralmente uma quilometragem ou um tempo de viagem. Esse problema real é contraído em um VRP mantendo os nós do depósito e do cliente e calculando os custos do caminho mais curto c_{ij} usando o algoritmo de Dijkstra, por exemplo (consulte [1] para obter uma implementação eficiente).

O VRP é NP-hard e algumas instâncias euclidianas com 75 nós ainda não foram resolvidas de forma otimizada [2]. Portanto, são necessários algoritmos heurísticos para lidar com instâncias da vida real. Os métodos mais simples e rápidos são as heurísticas construtivas simples, como as clássicas propostas por Clarke e Wright, Mole e Jameson, e Gillett e Miller: veja Laporte et al. [3] para uma pesquisa recente estendida aos algoritmos de busca tabu, e Christofides et al. [4], para uma revisão anterior. As metaheurísticas oferecem soluções muito melhores, especialmente em instâncias de grande escala. Dois excelentes levantamentos para essa área de pesquisa muito ativa são fornecidos por Gendreau et al. [5] e por Golden et al. [6]. Eles mostram que as melhores metaheurísticas para o VRP são os poderosos algoritmos de busca tabu (TS), que superam facilmente outras metaheurísticas, como recozimento simulado (SA), algoritmos genéticos (GA) e algoritmos de formigas.

Com relação aos algoritmos genéticos, vale a pena citar Van Breedam e Schmitt. Van Breedam [7] mede o impacto de vários parâmetros em um AG e um SA e compara os dois algoritmos com um TS desenvolvido anteriormente. Seus cromossomos contêm uma substring por viagem e várias cópias do depósito usadas como delimitadores de viagem. O crossover amplia o crossover clássico baseado em ordem PMX
[8] e pode produzir filhos inviáveis que são rejeitados. Van Breedam relata resultados comparáveis para os três algoritmos, mas o estudo é realizado em seu próprio conjunto de instâncias e não há comparação com metaheurísticas já avaliadas na literatura.

Schmitt [9,10] projetou um AG com cromossomos de permutação do tipo TSP (ou seja, sem delimitadores de viagem). A aptidão é calculada por meio da varredura dos cromossomos a partir do primeiro cliente: uma nova viagem é criada toda vez que o próximo cliente não se encaixa na viagem atual. Usando o cruzamento OX [11], Schmitt avalia seu AG nas 14 instâncias clássicas propostas por Christofides et al. [4] e obtém resultados decepcionantes: o AG supera facilmente a heurística de Clarke e Wright, mas não consegue competir com algumas heurísticas construtivas combinadas com procedimentos de melhoria de descida mais íngreme.

De fato, vários autores, como Gendreau [5], destacam a falta de um AG competitivo para o VRP. Essa situação parece anormal, pois existem AGs eficientes disponíveis tanto para um problema mais simples, como o Problema do Caixeiro Viajante ou TSP [12], quanto para uma extensão, como o VRP com janelas de tempo ou VRPTW [13,14].

O objetivo deste artigo é remediar essa anomalia apresentando um AG híbrido relativamente simples, mas e$caz, para uma versão do VRP especificada na Seção 2. A Seção 3 destaca alguns pontos-chave essenciais para a eficiência do AG. Os cromossomos e sua avaliação são descritos na Seção 4, enquanto as Seções 5, 6, 7 e 8 são, respectivamente, dedicadas ao cruzamento, à busca local usada como operador de mutação, à estrutura da população e às iterações do AG. Uma avaliação computacional é apresentada na Seção 9.

2. Problema tratado e suposições

Os benchmarks clássicos usados na Seção 9 para testes compreendem algumas instâncias de *VRP com restrição de distância* (DVRP): cada cliente está associado a um custo de entrega não negativo d_i e o custo total de cada viagem (custos de viagem mais custos de entrega) não pode exceder um limite fixo L. É por isso que o nosso GA foi projetado para o DVRP. É claro que ele pode resolver o VRP puro definindo todos os d_i como 0 e L como um valor enorme.

$$\forall i, j, k \in V : c_{ik} + c_{kj} \geq c_{ij}, \tag{1}$$

$$\forall i \in V \setminus \{0\} : q_i \leq W, \tag{2}$$

$$\forall i \in V \setminus \{0\} : c_{0i} + d_i + c_{i0} \leq L. \tag{3}$$

Na sequência, o número de veículos m é uma variável de decisão, os custos são números reais e a matriz de custos satisfaz a desigualdade triangular (1). Para garantir a existência de soluções viáveis, os dados de entrada satisfazem a condição (2) (nenhuma demanda deve exceder a capacidade do veículo) e a condição (3) (uma viagem de retorno para cada cliente deve ser possível dentro do limite L). O objetivo é minimizar o custo total das viagens. Como veremos, o AG é flexível o suficiente para lidar com objetivos mais complicados.

3. Esboço dos pontos-chave no projeto do GA

Nosso GA para o DVRP baseia-se nas seguintes ideias-chave, desenvolvidas nas seções a seguir:

K1. Cromossomos de permutação do tipo TSP, sem delimitadores de viagem. K2. Cálculo exato da aptidão usando um procedimento de divisão.
K3. População pequena na qual todos os indivíduos são distintos.
K4. Inclusão de boas soluções heurísticas na população inicial. K5. Procedimento de aprimoramento como operador de mutação (GA híbrido).
K6. Gerenciamento incremental da população.
K7. Fase de exploração principal seguida de algumas reinicializações curtas.

Os pontos 3 a 7 já foram combinados por Prins em um AG muito eficiente para o problema de programação de lojas abertas [15]. Esse AG supera dois outros AGs e três algoritmos TS. Ele ainda é o único algoritmo capaz de resolver de forma otimizada 30 instâncias com 100, 225 e 400 tarefas propostas por Taillard, exceto uma para a qual a solução mais conhecida foi aprimorada. Mais recentemente, Lacomme et al. [16] aplicaram as mesmas ideias, mais 1-2 ideias específicas para roteamento de veículos, ao problema de roteamento de arco capacitado (CARP). Novamente, o AG resultante tem um desempenho muito bom em problemas clássicos de referência: dois terços são resolvidos de forma otimizada e oito soluções mais conhecidas são comprovadas. A falta de um AG competitivo foi a principal motivação para experimentar essas ideias bem-sucedidas no DVRP.

4. Cromossomos e avaliação

4.1. Princípios

Como na maioria dos GAs para o TSP, um cromossomo é simplesmente uma sequência (permutação) S de n nós clientes, *sem delimitadores de viagem*. Ele pode ser interpretado como a ordem em que um veículo deve visitar todos os clientes, se o mesmo veículo realizar todas as viagens uma a uma. Ao contrário de Schmitt, que corta S em viagens sequencialmente, usamos um *procedimento de divisão ideal Split*, detalhado em 4.2, para obter a melhor solução DVRP respeitando a sequência. A adequação $F(S)$ de S é o custo total dessa solução.

Esse tipo de método foi apresentado pela primeira vez por Beasley [17] como a segunda fase de uma heurística route-first, cluster-second para o VRP. A primeira fase calcula uma rota gigante do TSP em todos os clientes, relaxando a capacidade do veículo e o comprimento máximo da rota. Além do interesse teórico em provar algumas relações de desempenho no pior caso, o método nunca substituiu a heurística VRP mais tradicional.

A situação é diferente em um AG. Primeiro, *existe pelo menos um cromossomo ideal* (considere qualquer solução DVRP ideal e concatene as listas de nós de suas viagens). Segundo, *se um cruzamento gerar esse cromossomo*, *a solução DVRP ideal correspondente sempre poderá ser recuperada com Split*. Terceiro, *a tarefa de encontrar o melhor cromossomo é deixada para o paralelismo intrínseco do AG*.

4.2. Algoritmo

O Split funciona em um gráfico auxiliar $H = (X, A, Z)$. X contém $n + 1$ nós indexados de 0 a n. A contém um arco (i, j), $i < j$, se uma viagem que visite os clientes S_{i+1} a S_j for viável em termos de carga (condição (4)) e custo (condição 5)). O peso z_{ij} de (i, j) é igual ao custo da viagem.

$$\delta(i,j) \in A : q \sum_{k=i+1}^{j} d_{S_k} \le W, \tag{4}$$

$$\delta(i,j) \in A : z_{ij} = c_{0,S_{i+1}} + \sum_{k=i+1}^{j} (d_{S_k} + c_{S_k,S_{k+1}}) + d_{S_j} + c_{S_j,0} \le L. \tag{5}$$

Uma solução DVRP ideal para S corresponde a um caminho de custo mínimo μ de 0 a n em H. Essa avaliação é razoavelmente rápida porque H não tem circuitos, $|A| = O(n^2)$, e a numeração dos nós fornece uma ordenação topológica natural: nesse caso, μ pode ser calculado em $O(n^2)$ usando o algoritmo de Bellman [1]. O algoritmo é mais rápido quando a demanda mínima $q_{\min}$ é grande o suficiente: como uma viagem não pode visitar mais de $b = \lfloor W/q_{\min} \rfloor$ clientes, a complexidade se torna $O(nb)$.

A parte superior da Fig. 1 mostra uma sequência $S = (a, b, c, d, e)$ com $W = 10$, $L = \infty$ e demandas entre parênteses. H no meio contém, por exemplo, um arco ab com peso 55 para a viagem $(0, a, b, 0)$. μ tem três arcos e seu custo é 255 (linhas em negrito). A parte inferior apresenta a solução VRP resultante com três viagens.

O algoritmo da Fig. 2 é uma versão no espaço $O(n)$ que não gera H explicitamente. Ele calcula dois rótulos para cada nó $j = 1, 2, \ldots, n$ de X: V_j, o custo do caminho mais curto do nó 0 ao nó j em H, e P_j, o predecessor de j nesse caminho. O loop *de repetição* enumera todas as subseqüências viáveis $S_i \ldots S_j$ e atualiza diretamente V_j e P_j. A adequação necessária $F(S)$ é dada no final por V_n. Para um determinado i, observe que a incrementação de j para quando L é excedido: nenhuma viagem viável é descartada, pois a desigualdade triangular é válida.

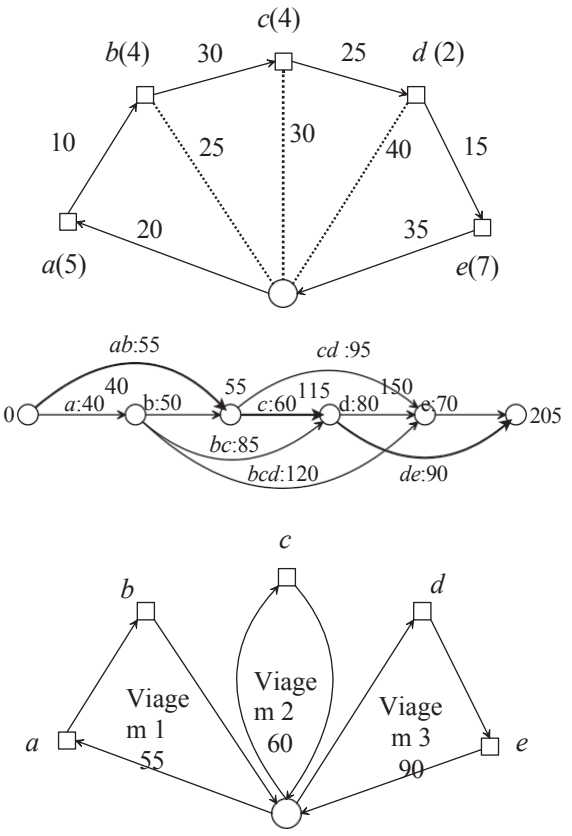

Fig. 1. Exemplo de avaliação de cromossomos.

O vetor de rótulos P é mantido com o cromossomo para extrair a solução DVRP no final do AG, usando o algoritmo da Fig. 3. Ele cria até n viagens (no pior caso, com um veículo por demanda). Cada viagem é uma lista de clientes, possivelmente vazia. O procedimento *enqueue* adiciona um nó no final de uma viagem. O número de viagens (ou veículos) não vazias efetivamente usadas é dado por t.

4.3. Extensões para outras funções objetivas

O Split é muito flexível. Os três exemplos a seguir mostram como lidar com funções objetivas mais complicadas. A complexidade de tempo $O(n^2)$ é mantida nos dois primeiros exemplos.

Custo total e número de veículos. O algoritmo da Fig. 2 pode ser adaptado para quebrar empates no custo com o número de veículos necessários: (a) definir para cada nó um segundo rótulo N_i para o número de arcos no caminho mais curto, (b) definir N_0 como 0 no início, (c) no caso de $V_{i-1} + custo < V_j$, adicionar $N_j := N_{i-1} + 1$, (d) adicionar um caso $V_{i-1} + custo = V_j$ que atualiza P_j e N_j se $N_{i-1} + 1 < N_j$. Também é possível minimizar primeiro o número de veículos, testando N_j antes de V_j.

Problema de combinação de frota de veículos (*VFMP*): no VFMP, a frota ainda não foi comprada. Vários tipos de veículos são possíveis, com determinadas capacidades e preços. O objetivo é encontrar uma composição de frota

```
V_0 := 0
for i := 1 to n do V_i := +∞ endfor
for i := 1 to n do
   load := 0; cost := 0; j := i
   repeat
      load := load + q_{S_j}
      if i = j then
         cost := c_{0,S_j} + d_{S_j} + c_{S_j,0}
      else
         cost := cost − c_{S_{j−1},0} + c_{S_{j−1},S_j} + d_{S_j} + c_{S_j,0}
      endif
      if (load ≤ W) and (cost ≤ L) then
         //here substring S_i ... S_j corresponds to arc (i − 1, j) in H
         if V_{i−1} + cost < V_j then
            V_j := V_{i−1} + cost
            P_j := i − 1
         endif
         j := j + 1
      endif
   until (j > n) or (load > W) or (cost > L)
enfor.
```

Fig. 2. Algoritmo para o procedimento de divisão *split*.

```
for i := 1 to n do trip(i) := ∅ endfor
t := 0
j := n
repeat
   t := t + 1
   i := P_j
   for k := i + 1 to j do enqueue(trip(t), S_k) endfor
   j := i
until i = 0.
```

Fig. 3. Algoritmo para extrair a solução VRP do vetor *P*.

minimizando o custo operacional (custo total das viagens) mais o custo da frota. Para adaptar o algoritmo, ao examinar uma viagem S_i - - - S_j, basta adicionar ao *custo* o custo do veículo mais barato com capacidade não inferior à *carga*.

Número limitado de veículos: O PMFV corresponde a uma frota heterogênea com um número ilimitado de veículos para cada tipo. Também é possível lidar com um número limitado de veículos idênticos (frota homogênea) ou uma frota heterogênea com um número limitado de veículos para cada tipo. O último problema corresponde a um caminho mais curto com restrição de recursos em H. Embora seja NP-difícil, ele pode ser resolvido em tempo pseudopolinomial [18].

```
Rank:  1   2   3   4   5   6   7   8   9
                   i=4     j=6
                    ↓       ↓
P1  :  1   3   2 | 6   4   5 | 9   7   8
P2  :  3   7   8 | 1   4   9 | 2   5   6

C1  :  8   1   9   6   4   5   2   3   7
C2  :  2   6   5   1   4   9   7   8   3
```

Fig. 4. Exemplo de crossover OX.

5. Crossover

Nossos cromossomos sem delimitadores de viagem podem passar por cruzamentos clássicos, como o *cruzamento de ordem* OX ou o *cruzamento de ordem linear* LOX. Tradicionalmente, o LOX é usado quando os cromossomos têm claramente duas extremidades (por exemplo, quando codificam um caminho hamiltoniano ou uma programação de uma máquina), enquanto o OX é mais adequado para permutações cíclicas, como no TSP. Como uma solução DVRP pode fornecer cromossomos diferentes, dependendo da ordem de concatenação de suas viagens, não há nenhuma razão especial para distinguir uma "primeira" ou "última" viagem. É por isso que selecionamos o OX, depois de alguns testes que confirmaram sua superioridade em relação ao LOX.

O exemplo na Fig. 4 mostra como a OX constrói o primeiro filho C1. Primeiro, dois locais de corte i e j são selecionados aleatoriamente em $P1$, aqui $i = 4$ e $j = 6$. Em seguida, a substring $P1(i)$ - - - $P1(j)$ é copiada para C1(i) - - - C1(j). Por fim, $P2$ é varrido circularmente de $j + 1$ em diante para completar C1 com os nós ausentes. C1 também é preenchido circularmente a partir de $j + 1$. O outro filho C2 pode ser obtido pela troca das funções de P1 e $P2$. Todo o procedimento pode ser implementado em $O(n)$.

6. Pesquisa local como operador de mutação

Para competir com a heurística TS ou SA, a estrutura clássica do AG deve ser hibridizada com algum tipo de procedimento de aprimoramento, resultando em um *AG híbrido* ou *algoritmo memético* [19]. Para o DVRP, obtivemos rapidamente resultados muito melhores ao substituir operadores de mutação simples (como mover ou trocar alguns nós) por um procedimento de busca local LS. Isso não leva a uma convergência prematura, graças à técnica de dispersão explicada na Seção 7.

Um filho C produzido por OX é aprimorado por LS com uma probabilidade fixa p_m . A sequência de nós de C é primeiro convertida em uma solução DVRP usando o algoritmo 3. Em seguida, o LS varre as vizinhanças $O(n^2)$ listadas abaixo. As vizinhanças mais amplas aumentam o tempo de execução do GA sem melhorar o resultado final. As vizinhanças pequenas são suficientes porque são compensadas pelo paralelismo intrínseco do AG: de fato, as soluções que podem ser alcançadas por cruzamentos de uma determinada população definem um tipo de vizinhança ampla que traz diversificação suficiente.

Cada iteração do LS varre em $O(n^2)$ todos os pares possíveis de nós distintos (u, v). Esses nós podem pertencer à mesma viagem ou a viagens diferentes e um deles pode ser o depósito. Para cada par, os seguintes movimentos simples são testados. x e y são os sucessores de u e v em suas respectivas viagens.

$T(u)$ é a viagem que visita u.

M1. Se u for um nó cliente, *remova-o* e insira-o após v,
M2. Se u e x forem clientes, remova-os e insira (u, x) depois de v,
M3. Se u e x forem clientes, remova-os e insira (x, u) depois de v,
M4. Se u e v forem clientes, troque u e v,
M5. Se u, x e v forem clientes, troque (u, x) e v,
M6. Se (u, x) e (v, y) forem clientes, troque (u, x) e (v, y),
M7. Se $T(u) = T(v)$, substitua (u, x) e (v, y) por (u, v) e (x, y),
M8. Se $T(u) /= T(v)$, substitua (u, x) e (v, y) por (u, v) e (x, y),
M9. Se $T(u) /= T(v)$, substitua (u, x) e (v, y) por (u, y) e (x, v).

O movimento 7 corresponde ao conhecido movimento 2-opt, enquanto os movimentos 8-9 estendem o 2-opt para viagens diferentes. Cada iteração do LS para no primeiro movimento de melhoria. A solução é então modificada e o processo é repetido até que não seja possível encontrar mais nenhuma economia. Movimentos como o M1 podem esvaziar uma viagem (que se torna um simples loop no depósito), mas também podem atribuir novos clientes a ela mais tarde. É por isso que as viagens vazias são removidas somente no final do LS. A solução final com um custo $\beta \leq F(C)$ é convertida em um cromossomo mutante M pela concatenação de suas viagens. *A divisão* é aplicada em todos os casos a M porque, às vezes, ela encontra uma partição melhor em viagens para a mesma sequência (observe que $F(M) \leq \beta \leq F(C)$).

7. Estrutura e inicialização da população

A população é implementada como uma matriz W de σ cromossomos, sempre classificados em ordem crescente de custo para facilitar a iteração básica do AG descrita na Seção 8. Portanto, a melhor solução é W_1 . Para evitar recomputações inúteis, a aptidão calculada por *Split* é mantida em cada cromossomo. Isso pode ser feito codificando cada cromossomo como um registro com dois campos, uma sequência de nós e um valor de adequação.

Os clones (soluções idênticas) são proibidos em W para garantir uma melhor dispersão das soluções e diminuir o risco de convergência prematura. Isso também permite uma taxa de mutação mais alta p_m pela pesquisa local, proporcionando um AG mais agressivo. Para evitar a comparação de cromossomos em detalhes e acelerar a detecção de clones, impomos uma condição mais rigorosa: os custos de quaisquer duas soluções devem ser espaçados pelo menos por uma constante $\Delta > 0$, como na relação (6). Também tentamos uma detecção exata de clones, sem observar uma melhoria significativa. Na sequência, uma população que satisfaça a condição 6 será considerada *bem espaçada*. A forma mais simples com $\Delta = 1$ (soluções com custos inteiros distintos) já foi usada em um AG eficiente para o problema de programação de lojas abertas [15].

$$\forall P_1, P_2 \in W : P_1 /= P_2 \Rightarrow |F(P_1) - F(P_2)| \geq \Delta \tag{6}$$

No início, três boas soluções são calculadas usando a heurística de Clarke e Wright [20] (versão paralela na qual cada fusão é seguida por uma busca local de 3 opções), Mole e Jameson [21] e Gillett e Miller [22]. Essas heurísticas são denominadas CW, MJ e GM na sequência. As viagens de cada solução são concatenadas em um cromossomo. Como após uma mutação por busca local, *a divisão* deve ser aplicada para definir a verdadeira aptidão, que às vezes é um pouco melhor do que o custo encontrado pela heurística. Os cromossomos resultantes são armazenados em W_1 , W_2 e W_3 .

Em seguida, para $k = 4, 5, \ldots, \sigma$, cada outro cromossomo W_k é inicializado como uma permutação aleatória de clientes, avaliada por *Split*. Como um sorteio pode gerar um clone, tentamos até τ vezes (τ=50 normalmente) para obter um W_k aceitável, ou seja, de modo que o conjunto W_1 - - - W_k seja bem espaçado. Se não for possível obter o cromossomo atual W_k , o tamanho da população será truncado para $\sigma = k$ - 1 cromossomos. Esperamos que isso ocorra somente quando n for muito pequeno (por exemplo, 10 clientes) ou quando σ for muito grande. A população é finalmente classificada em ordem crescente de custo.

A partir de uma população vazia, a inicialização adiciona um cromossomo de cada vez. Posteriormente, em cada iteração do GA, apenas um cromossomo é substituído (consulte a Seção 8). Portanto, o espaçamento pode ser verificado em cada etapa em $O(1)$ da seguinte forma. Observe que dois cromossomos P_1 e P_2 estão bem espaçados se seus *custos escalonados* $\lfloor F(P_1)/\Delta \rfloor$ e $\lfloor F(P_2)/\Delta \rfloor$ forem diferentes. Defina um vetor 0 -1 U , indexado de 0 em diante, de modo que $U(\varphi) = 1$ i$ exista em W uma solução com um custo escalonado φ. Se $F_{\max}$ for um limite superior para a aptidão, U poderá ser dimensionado até o índice $\lfloor F_{\max}/\Delta \rfloor$. No início, W é vazio e U é inicializado em 0. Um novo cromossomo K pode ser adicionado a W i$ $U(\lfloor F(K)/\Delta \rfloor) = 0$. Depois de adicionado, $U(\lfloor F(K)/\Delta \rfloor)$ deve ser definido como 1.

8. Iterações do GA e estrutura geral resultante

8.1. Descrição da iteração básica

Os pais são selecionados com o *método de torneio binário*. Dois cromossomos são selecionados aleatoriamente da população e o de menor custo torna-se o primeiro pai *P1*. O procedimento é repetido para obter o segundo pai *P2*. OX é aplicado para gerar dois filhos C1 e C2. Um filho C é selecionado aleatoriamente para substituir um cromossomo medíocre W_k desenhado acima da mediana, ou seja, com $k \geq \lfloor \sigma/2 \rfloor$. Esse *gerenciamento incremental da população* preserva a melhor solução e permite que um bom filho se reproduza rapidamente. Se $(W \setminus \{W_k\}) \cup \{C\}$ estiver bem espaçado, a substituição será aceita: W é reordenado em $O(\sigma)$ movendo C para uma posição ordenada, e uma *iteração produtiva* (ou seja, bem-sucedida) é contada.

Uma taxa de rejeição excessiva (ou seja, a maioria das iterações do AG é improdutiva) aparece se Δ ou o tamanho da população σ forem muito grandes. Esse fenômeno piora com uma alta taxa de mutação p_m porque a busca local LS comprime a população em um intervalo de custo menor. Na prática, uma taxa de rejeição razoável (no máximo 10% das iterações) é obtida com $\sigma \leq 50$, $0{,}2 \leq \Delta \leq 5$ e $p_m \leq 0{,}3$.

C é selecionado aleatoriamente porque o desempenho médio do GA é ligeiramente inferior quando W_k é substituído pelo melhor filho ou se duas soluções são substituídas por C1 e C2. Quando C sofre mutação, o mutante é construído em um cromossomo separado M . Se $(W \setminus \{W_k\}) \cup \{M\}$ estiver bem espaçado, M é atribuído à variável C. O objetivo desse truque é tentar uma substituição por C se seu mutante for rejeitado (isso fica claro no algoritmo da Fig. 5). Isso também diminui a taxa de rejeição e melhora ligeiramente a solução final em média.

8.2. Fase principal e reinicializações

O AG executa uma fase principal, parando após um número máximo de iterações produtivas, $um_{\max}$ (número de cruzamentos que não geram um clone), após um número máximo de iterações sem

melhorar a melhor solução, $\beta_{\max}$, ou quando atinge o custo ideal ou um limite inferior (LB)

```
//initial heuristic solutions
Π1 := solution of heuristic CW
Π2 := solution of heuristic MJ
if not spaced({Π1, Π2}) then k := 1 else k := 2 endif
Πk+1 := solution of heuristic GM
if spaced({Π1 ... Πk+1}) then k := k + 1 endif
//initial random solutions
repeat
   k := k + 1
   try := 0
   //try up to τ times to get Πk
   repeat
      try := try + 1
      generate Πk at random
   until spaced({Π1 ... Πk}) or (try > τ)
until (k = σ) or (try > τ)
if try > τ then σ := k − 1 endif
sort Π in increasing cost order
//main loop of GA
α, β := 0
//iterations and iterations without improvement
repeat
   select two parents P1 and P2 by binary tournament
   apply OX to (P1, P2) and select one child C at random
   select k at random in [⌊σ/2⌋, σ]
   if random < pm then //mutation
      M := C mutated with the local search LS
      //if M is a clone, replacement attempted with C
      if spaced((Π \ {Πk}) ∪ {M}) then C := M endif
   endif
   if spaced((Π \ {Πk}) ∪ {C}) then //productive iteration
      α := α + 1
      if F(C) < F(Π1) then β := 0 else β := β + 1 endif
      Πk := C
      shift Πk to re-sort Π
   endif
until (α = αmax) or (β = βmax) or (F(Π1) = LB).
```

Fig. 5. Algoritmo geral para o AG.

conhecido para algumas instâncias (nesse caso, W_1 é, obviamente, ideal). Essa fase principal está resumida na Fig. 5, na qual *spaced* (A) é uma função booleana verdadeira se o conjunto de cromossomos A estiver bem espaçado. Algumas instâncias difíceis são melhoradas com algumas reinicializações baseadas em um *procedimento de substituição parcial* proposto por Cheung et al. [23] em um AG para um problema de localização de instalações. Esse procedimento deve ser

adaptado para populações bem espaçadas. Seja ρ o número de cromossomos a serem substituídos, normalmente $\sigma/4$. Gere aleatoriamente uma população bem espaçada Ω de ρ cromossomos, de modo que $W \cup \{\Omega_k\}$ seja bem espaçada para $k = 1, 2, \ldots \rho$. Para $k = 1, 2, \ldots \rho$, se $F(\Omega_k) < F(W_\sigma)$ (ou seja, se o novo cromossomo for melhor do que a pior solução em W), substitua W_σ por Ω_k. Caso contrário, cruze Ω_k com todos os cromossomos em $W \cup \Omega$, mantenha o melhor filho C obtido por esses cruzamentos e substitua W_σ por C se $F(C) < F(W_{nc})$.

Na prática, várias populações Ω devem ser testadas para obter ρ substituições bem-sucedidas. Limitamos cada reinício do a cinco tentativas (mesmo que menos de ρ soluções sejam substituídas), pois o procedimento de substituição parcial pode consumir muito tempo em instâncias grandes. Em comparação com uma substituição cega, o procedimento de substituição parcial preserva a melhor solução e não pode aumentar o pior custo. Os experimentos numéricos também indicam uma descida mais rápida do AG.

9. Avaliação computacional

9.1. Implementação e benchmarks

O AG foi implementado na linguagem Pascal Delphi 5, em um PC Pentium-3 com clock de 1 GHz e sistema operacional Windows 98. A avaliação é realizada em dois conjuntos de problemas de benchmark *euclidiano*, ou seja, os vértices são definidos por pontos no plano e o custo de cada borda (i, j) é a distância euclidiana entre i e j.

O primeiro conjunto contém 14 problemas clássicos propostos por Christofides et al. [4] e pode ser baixado da Biblioteca OR [24]. Esses arquivos têm de 50 a 199 clientes. Os únicos com restrição de comprimento de rota (DVRPs) são os arquivos 6-10, 13 e 14, todos com tempos de serviço diferentes de zero. Os outros são VRPs puros. As soluções calculadas por vários algoritmos TS e SA são fornecidas por Gendreau et al. [5] e Golden et al. [6].

O segundo conjunto contém 20 instâncias de grande escala (200 a 483 clientes) apresentadas por Golden et al. [6], consulte [25] para obter um endereço na Internet. As instâncias 1-8 são DVRPs, mas com tempos de serviço nulos. Golden et al. descrevem em seu artigo um algoritmo de recozimento determinístico chamado *RTR* (*record-to-record travel*) e listam os custos da solução obtidos por esse método e por uma heurística TS projetada por Xu e Kelly [26], chamada XK na sequência.

Além do nosso AG, o único algoritmo que foi avaliado em ambos os conjuntos é um novo método promissor chamado de *busca tabu granular* (*GTS*) e proposto por Toth e Vigo [27]. Ele se baseia na observação de que as bordas mais longas de uma rede raramente são incluídas nas soluções ideais. Esse método descarta todas as bordas cujo comprimento excede um limite de granularidade, ajustado dinamicamente durante o algoritmo.

9.2. Parâmetros comuns do GA

As ideias-chave listadas na Seção 3 são essenciais para o bom desempenho do GA. Realizamos um longo estudo que mostra uma degradação moderada (+) ou forte (++) dos valores médios da solução se apenas uma das seguintes alterações for tentada:

C1. Tamanho da população $\sigma < 25$ ou $\sigma > 50$(+),
C2. Nenhuma solução inicial boa em W, apenas uma ou mais de 5(+),

C3. Pesquisa local LS aplicada a cada solução inicial de $W(+)$,
C4. Clones permitidos na população (nenhum teste de espaçamento) (++), C5. Seleção dos pais por um método de roleta (+),
C6. Mutação por um simples movimento ou troca de nós em vez de LS (++), C7. Gerenciamento de população geracional (+),
C8. Substituição pelo melhor filho de OX ou por dois filhos (+), C9. Substituir a pior solução, não uma acima da mediana (+).

Se todas as ideias-chave forem mantidas, o AG terá melhor desempenho com populações pequenas de $\sigma = 30$ soluções, $\Delta = 0{,}5$ e uma taxa de mutação relativamente alta p_m (até 0,20). σ e p_m são pequenos o suficiente para evitar a perda de muito tempo em cruzamentos improdutivos. Cada reinício substitui $\rho = 8$ soluções. O valor exato de p_m e o equilíbrio entre a fase principal e as reinicializações dependem dos conjuntos de instâncias e são especificados nas subseções correspondentes.

Após uma discussão com Rafael MartTi, da Universidade de Valência (Espanha), nosso AG obviamente tem algumas conexões com a *busca por dispersão* [28]: populações pequenas, alta taxa de mutação, reinícios correspondentes às fases curtas da busca por dispersão. Entretanto, nosso algoritmo gera filhos de forma estocástica e seu desempenho diminui se a busca local for aplicada a cada solução inicial. Esse último ponto é provavelmente o mais surpreendente. Ele não segue o modelo geral de algoritmo memético proposto por Moscato [19].

9.3. Resultados em instâncias Christo9des

A melhor configuração padrão dos parâmetros inclui os parâmetros comuns ($\sigma = 30$, $\Delta = 0{,}5$, $\rho = 8$), concluídos da seguinte forma. O AG executa a fase principal com $p_m = 0{,}05$, terminando após $um_{\max} = 30.000$ cruzamentos produtivos, $\beta_{\max} = 10.000$ cruzamentos sem melhorar a melhor solução ou quando o ótimo é alcançado (é conhecido apenas para os problemas 1 e 12). Se o ótimo não for alcançado, o AG é reiniciado por 10 fases curtas com $um_{\max} = \beta_{\max} = 2000$ e p_m aumentado para 0,1. Os resultados são apresentados na Tabela 1.

As três primeiras colunas fornecem o número do problema, o número de clientes n e o valor da melhor solução conhecida (BKS) relatado em [5,6]. O melhor custo de solução $F(W_1)$ na população inicial é mostrado na coluna 4 e é sempre obtido por uma das três heurísticas CW, MJ e GM. As colunas 5 a 8 referem-se ao GA no qual os parâmetros padrão são aplicados a todas as instâncias. As colunas 5 (GA 3000) e 6 apresentam o custo da solução e o tempo de CPU usado em segundos para 3.000 cruzamentos (aproximadamente um décimo da fase principal). As colunas 7 (GA) e 8 fornecem as mesmas informações para a melhor solução encontrada. A coluna 9 (Melhor GA) apresenta o melhor resultado que encontramos até agora com várias configurações de parâmetros. A última linha apresenta os desvios médios em % das soluções mais conhecidas e os tempos médios de CPU.

Quando o AG é iniciado, os desvios médios das heurísticas CW, MJ e GM em relação às soluções mais conhecidas são, respectivamente, 6,81, 10,76 e 12,78%. Graças a um efeito de compensação, o desvio se torna 4,77% para a melhor das três soluções. Usando uma configuração, o AG para com um desvio médio de 0,23% e recupera oito soluções mais conhecidas (negrito). Usando várias configurações, ele chega a 10 soluções mais conhecidas com um desvio reduzido para 0,08%. Observe que a maioria das soluções mais conhecidas provavelmente é ótima: elas permanecem estáveis desde os últimos aprimoramentos publicados em 1995 para os problemas 5 e 10.

Tabela 1
Resultados da AG para instâncias da Christofides

Pb	*n*	BKS	$F_{(W1)}$	GA 3000	Tempo	GA	Tempo	Melhor GA
1	50	∗*524*.61	531.90	524.61	0.50	524.61	0.50	524.61
2	75	835.26	902.04	850.32	5.06	835.26	46.36	835.26
3	100	826.14	878.35	833.50	14.01	826.14	27.63	826.14
4	150	1028.42	1078.54	1049.24	42.79	1031.63	330.11	1030.46
5	199	1291.45	1383.42	1330.26	107.22	1300.23	1146.52	1296.39
6	50	555.43	584.06	555.43	3.40	555.43	0.66	555.43
7	75	909.68	956.23	921.42	9.72	912.30	85.18	909.68
8	100	865.94	906.22	866.87	18.45	865.94	22.41	865.94
9	150	1162.55	1280.90	1165.68	80.19	1164.25	434.90	1162.55
10	199	1395.85	1531.98	1434.52	177.91	1420.20	1609.87	1402.75
11	120	1042.11	1049.43	1042.11	23.57	1042.11	17.85	1042.11
12	100	∗*819*.56	824.42	819.56	2.70	819.56	2.70	819.56
13	120	1541.14	1585.97	1548.58	46.80	1542.97	626.54	1542.86
14	100	866.37	868.50	866.37	15.76	866.37	5.16	866.37
Média			+ 4.79	+ 0.90	39.15	+ 0.23	311.17	+ 0.08

Os asteriscos indicam os ótimos comprovados. Os caracteres em negrito indicam as soluções mais conhecidas recuperadas. As médias nas colunas 4, 5, 7 e 9 são desvios das soluções mais conhecidas em %. Tempos de CPU em segundos em um PC Pentium-3 de 1 GHz.

A duração média do GA é de 5,2 minutos (até 26,8 minutos), mas nossa implementação não é especialmente otimizada e os tempos de execução podem ser melhorados. Por exemplo, listas de vizinhos mais próximos poderiam ser usadas para evitar a varredura completa de cada vizinhança. De qualquer forma, o AG limitado a 3.000 cruzamentos já é uma heurística muito boa, com cinco soluções mais conhecidas recuperadas (incluindo as duas ótimas), um desvio médio de 0,9% e um tempo de execução muito melhor, de 39 s. Observe que esse AG truncado recupera 80% da economia obtida pelo AG completo, em 13% do seu tempo de execução.

A comparação com os algoritmos TS pesquisados em [5,6] é difícil, porque os autores raramente fornecem suas soluções mais conhecidas para uma configuração padrão de parâmetros, como recomendado atualmente. Em geral, eles fornecem as melhores soluções obtidas com configurações diferentes e não mencionam os tempos de CPU correspondentes.

No entanto, na Tabela 2, tentamos comparar nosso AG com 11 TS e 2 SA para o VRP. Todas as referências a esses algoritmos podem ser encontradas em [5] ou [6]. Nossa tabela compila várias tabelas de Golden et al., além dos resultados obtidos pelo GTS de Toth e Vigo. Os critérios são a distância média em % das soluções mais conhecidas (coluna Adbks), o número de soluções mais conhecidas recuperadas (Nbks) e o tempo médio de computação em minutos. A tabela, classificada em ordem crescente de distância em relação às melhores soluções, mostra resultados muito animadores: apenas uma TS publicada entre 10 se sai melhor do que o AG. Mesmo com uma única configuração de parâmetro, o AG estaria na posição 4, com um desvio médio de 0,23%.

Com relação aos tempos de CPU, Osman usa um Vax 8600, Gendreau um Silicon Graphics de 36 MHz (cerca de 5,7 MFlops), Xu e Kelly um DEC Alpha não especificado (26-43 MFlops, cerca de 5 vezes mais rápido que o Silicon Graphics, de acordo com Gendreau) e Toth e Vigo um PC de 200 MHz (15 MFlops). O PC de 1 GHz que executa o GA tem uma potência estimada de 75 MFlops. A última coluna da Tabela 2 fornece

Tabela 2
Comparação da metaheurística VRP nas 14 instâncias de Christofides

Tipo	Autores	Ano	Adbks %	Nbks	Tempo médio	Escalonado
TS	Taillard	1993	0.05	12	-	-
GA	Prins	2001	0.08	10	5.2	1.46
TS	Xu-Kelly (sete problemas)	1996	0.10	5	103.0	11.00
TS	Gendreau et al.	1994	0.20	8	-	-
TS	Taillard	1992	0.39	6	-	-
TS	Rego e Roucairol	1996	0.55	6	-	-
GTS	Toth e Vigo	1998	0.55	4	3.5	0.20
TS	Gendreau et al.	1991	0.68	5	-	-
TS	Rego e Roucairol	1996	0.77	4	-	-
TS	Gendreau et al.	1994	0.86	5	46.8	1.00
GA	Prins (3000 Xovers)	2001	0.90	5	0.7	0.20
TS	Osman	1993	1.01	4	26.1	?
TS	Osman	1993	1.03	3	34.0	?
SA	Osman	1993	2.09	2	-	-

Tempos de CPU em minutos em vários computadores.

tempos escalonados: se T_a for a duração e P_a a potência do computador para um algoritmo a, seu tempo escalonado será $(T/T_{ag}) \times (P/P_{ag})$, com g representando a heurística TS de Gendreau. A velocidade notável do GTS deve ser destacada. O GA é 7 vezes mais lento que o GTS, mas 7,5 vezes mais rápido que a heurística TS de Xu e Kelly. A versão truncada do GA (3.000 cruzamentos) é tão rápida quanto o GTS e seus custos de solução ainda são honrosos.

9.4. Resultados em instâncias de grande escala

Os parâmetros comuns para essas instâncias incluem novamente $\sigma = 30$, $\Delta = 0{,}5$ e $\rho = 8$. Em vez de uma longa fase principal seguida de reinícios mais curtos (que consomem muito tempo nesses arquivos), o AG executa 10 fases com $\text{com}_{\max} = \beta_{\max} = 2000$ cruzamentos e com uma taxa de mutação relativamente alta $p_m = 0{,}2$. A Tabela 3 mostra os excelentes resultados obtidos.

Usando os mesmos parâmetros para todas as instâncias, o desvio médio das soluções mais conhecidas é de 5,82% quando o AG é iniciado, -0,20% com nove aprimoramentos após 2000 cruzamentos (uma fase) e
-0,78% com 11 melhorias no final. Mais precisamente, há uma diferença notável entre as instâncias de DVRP (arquivos 1-8) e os outros arquivos que contêm VRPs comuns. Todas as soluções mais conhecidas para instâncias de DVRP são aprimoradas, mesmo após 2000 cruzamentos, com um desvio final de -2,57%. Os VRPs básicos são aprimorados três vezes, com um custo final médio ligeiramente acima do custo mais conhecido (+0,41%). O preço médio a ser pago para resolver esses problemas de grande escala é quase uma hora de tempo de CPU.

A Tabela 4 compara os custos da solução obtidos com uma configuração de parâmetro pelo AG e os algoritmos RTR, XK e GTS mencionados na seção 9.1. Conforme destacado por Toth e Vigo [27], os melhores resultados relatados para o XK são baseados em distâncias truncadas. Se forem usados números reais, todas as soluções DVRP (arquivos 1-8) encontradas pelo XK e listadas por Golden et al. [6] se tornam inviáveis, o que explica por que elas são

Tabela 3
Resultados do GA para instâncias de grande escala com uma configuração de parâmetro

Pb	*n*	BKS	$F_{(W1)}$	GA 2000	CPU	GA	CPU
1	240	5736.15	5935.59	5662.43	12.43	5648.04	32.42
2	320	8553.03	8648.33	8488.15	30.85	8459.73	77.92
3	400	11402.75	11756.34	11038.62	57.40	11036.22	120.83
4	480	14336.36	14867.99	13743.74	85.47	13728.80	187.60
5	200	6680.36	6931.51	6460.98	4.22	6460.98	1.04
6	280	8712.76	8915.53	8412.90	13.16	8412.90	9.97
7	360	10515.33	10842.41	10268.37	37.97	10267.50	39.05
8	440	12036.24	12352.05	11872.23	76.10	11865.40	88.30
9	255	587.09	645.73	598.59	8.34	596.89	14.32
10	323	746.56	815.04	758.49	18.80	751.41	36.58
11	399	932.68	1004.04	952.15	28.15	939.74	78.50
12	483	1136.05	1225.50	1152.88	55.98	1152.88	30.87
13	252	868.70	949.39	881.98	6.56	877.71	15.30
14	320	1096.18	1205.37	1106.65	13.55	1089.93	34.07
15	396	1363.34	1503.23	1392.04	27.45	1371.61	110.48
16	480	1650.42	1818.49	1680.67	50.58	1650.94	130.97
17	240	709.90	759.58	717.09	7.67	717.09	5.86
18	300	1014.82	1059.28	1020.47	15.00	1018.74	39.33
19	360	1389.14	1452.89	1404.09	25.67	1385.60	74.25
20	420	1875.17	1928.61	1868.36	41.57	1846.55	210.42
Média			5.82	- 0.20	30.84	- 0.78	66.60

Os caracteres em negrito indicam que as soluções mais conhecidas foram aprimoradas. As médias nas colunas 4, 5 e 7 são desvios das soluções mais conhecidas em %. Tempos de CPU em minutos em um PC Pentium-3 de 1 GHz.

não incluídos na tabela. Além disso, os custos dos arquivos 17-20 aumentam (a tabela indica os valores corrigidos). Observe que os números reais de precisão dupla são absolutamente necessários para que haja duas casas decimais exatas no final de qualquer metaheurística.

O GA apresenta claramente os melhores resultados, com 12 melhores soluções em um total de 20. Os tempos de computação são fornecidos em minutos para um PC Pentium de 100 MHz a 8 MFlops (RTR), para um DEC Alpha a 26 MFlops (XK), para um PC Pentium de 200 MHz a 15 MFlops (GTS) e para um PC Pentium de 1 GHz a 75 MFlops (GA). Os tempos dimensionados usando o RTR como nível de referência mostram que o GA é muito mais lento que o GTS e o RTR, mas muito mais rápido que o XK.

Alguns comentários são necessários para fazer uma comparação justa entre o GTS e o GA. O GTS foi projetado para obter um excelente compromisso entre o custo da solução e o tempo de computação. Esse objetivo foi claramente alcançado: O GTS é, de longe, a metaheurística mais rápida para resolver o VRP. Além disso, sem o GA, o GTS teria 12 melhores soluções na tabela (problemas 2, 3, 5 a 8, 13, 14, 16 a 18). O objetivo do nosso trabalho não é um algoritmo rápido, mas sim trazer o GA eficaz que estava faltando para o VRP e definir novas soluções mais conhecidas, como um desafio para outras metaheurísticas.

As melhores soluções encontradas pelo AG e pelo GTS geralmente são aprimoradas se forem usadas várias combinações de parâmetros. A coluna 3 da Tabela 5 apresenta as soluções mais conhecidas antes do AG: 16 de 20 são obtidas pelo GTS (duas delas são fornecidas com uma casa decimal por Toth e Vigo). O AG

Tabela 4
Resultados de RTR, XK, GTS e GA em instâncias de grande escala (configuração de um parâmetro)

Pb	*n*	RTR	CPU	XK	CPU	GTS	CPU	GA	CPU
1	240	5834.60	3.68	-	802.87	5736.15	4.98	5648.04	32.42
2	320	9002.26	22.66	-	898.53	8553.03	8.28	8459.73	77.92
3	400	11879.95	40.04	-	1749.27	11402.75	12.94	11036.22	120.83
4	480	14639.32	122.61	-	2432.42	14910.62	15.13	13728.80	187.60
5	200	6702.73	11.24	-	591.40	6697.53	2.38	6460.98	1.04
6	280	9016.93	18.79	-	913.70	8963.32	4.65	8412.90	9.97
7	360	11213.31	22.55	-	1062.73	10547.44	11.66	10267.50	39.05
8	440	12514.20	111.37	-	1586.20	12036.24	11.08	1865.40	88.30
9	255	587.09	23.01	589.10	340.20	593.35	11.67	596.89	14.32
10	323	749.15	31.49	746.56	501.82	751.66	15.83	751.41	36.58
11	399	934.33	69.19	932.68	852.72	936.04	33.12	939.74	78.50
12	483	1137.18	101.09	1140.72	1151.10	1147.14	42.90	1152.88	30.87
13	252	881.04	6.01	881.07	1465.77	868.80	11.43	877.71	15.30
14	320	1103.69	21.83	1118.09	1577.30	1096.18	14.51	1089.93	34.07
15	396	1364.23	32.62	1377.79	4340.07	1369.44	18.45	1371.61	110.48
16	480	1657.93	47.55	1656.66	8943.45	1652.32	23.07	1650.94	130.97
17	240	720.44	5.69	747.24	2314.00	711.07	14.29	717.09	5.86
18	300	1029.21	8.15	1066.57	4101.02	1016.83	21.45	1018.74	39.33
19	360	1403.05	12.42	1435.88	5718.38	1400.96	30.06	1385.60	74.25
20	420	1875.17	31.05	1934.99	10839.73	1915.83	43.05	1846.55	210.42
Nbest		3		2		3		12	
Tempo			37.15		2609.13		17.54		66.60
Escalonado			1.00		228.24		0.88		16.80

As soluções em negrito indicam as melhores soluções entre RTR, XK, GTS e GA. Tempos de CPU em minutos.

encontra soluções melhores para todas as instâncias do DVRP (arquivos 1-8) e para quatro instâncias do VRP comum. A economia média é de 1,17%. Treze soluções mais conhecidas são agora fornecidas pelo GA (negrito), quatro pelo GTS, duas pelo XK e uma pelo RTR.

10. Conclusão

Este artigo apresenta o primeiro AG híbrido para o VRP capaz de competir com algoritmos TS poderosos em termos de custo médio de solução. Em instâncias Christofides, esse AG supera todas as metaheurísticas publicadas, exceto uma. Ele se torna o melhor algoritmo disponível para as instâncias de grande escala geradas por Golden et al. [6].

Esses resultados muito bons podem ser explicados por alguns recursos-chave. Uma possível convergência prematura devido à pesquisa local é evitada com o uso de pequenas populações de soluções distintas. Três heurísticas clássicas fornecem bons pontos de partida. O gerenciamento incremental da população e a técnica de substituição parcial usada nas reinicializações aceleram a diminuição da função objetiva.

Tabela 5
Novo status das soluções mais conhecidas para instâncias de grande escala

Pb	*n*	Melhor anterior		MétodoMelhor
solução GA				
1	240	5736.15	GTS	5646.63
2	320	8553.03	GTS	8447.92
3	400	11402.75	GTS	11036.22
4	480	14336.36	GTS	13624.52
5	200	6680.36	GTS	6460.98
6	280	8712.76	GTS	8412.80
7	360	10515.33	GTS	10195.59
8	440	12036.24	GTS	11828.78
9	255	587.09	RTR	591.54
10	323	746.56	XK	751.41
11	399	932.68	XK	933.04
12	483	1136.05	GTS	1133.79
13	252	868.7	GTS	875.16
14	320	1096.18	GTS	1086.24
15	396	1363.34	GTS	1367.37
16	480	1650.42	GTS	1650.94
17	240	709.9	GTS	710.42
18	300	1014.82	GTS	1014.80
19	360	1389.14	GTS	1376.49
20	420	1875.17	RTR	1846.55

O algoritmo de divisão que calcula a aptidão permite cromossomos e cruzamentos simples, como no TSP. Com exceção dessas ideias, o AG permanece relativamente simples.

Entretanto, um ponto precisa ser melhorado: o AG ainda é mais lento do que muitos algoritmos de TS, embora seja mais rápido do que alguns deles. Essa desvantagem se deve em parte à implementação atual, que poderia ser otimizada, e à busca local, baseada em vizinhanças básicas e que absorve 95% do tempo da CPU. Portanto, nosso principal objetivo agora é acelerar a busca local, sem sacrificar seu custo médio de solução. Também queremos desenvolver uma pesquisa de dispersão e compará-la com o GA, para quantificar o impacto da randomização em termos de custos de solução e tempos de CPU.

Referências

[1] Cormen TH, Leiserson CE, Rivest RL. Introduction to algorithms (Introdução aos algoritmos). Cambridge, MA: MIT Press, 1990.

[2] Toth P, Vigo D. Solução exata do problema de roteamento de veículos. Em: Crainic TG, Laporte G, editores. Fleet management and logistics. Dordrecht: Kluwer, 1998. p. 1-31.

[3] Laporte G, Gendreau M, Potvin JY, Semet F. Classical and modern heuristics for the vehicle routing problem (Heurística clássica e moderna para o problema de roteamento de veículos). International Transactions in Operational Research 2000;7:285-300.

[4] Christofides N, Mingozzi A, Toth P. O problema de roteamento de veículos. Em: Christofides N, Mingozzi A, Toth P, Sandi C, editores. Combinatorial optimization. New York: Wiley, 1979. p. 315-38.

[5] Gendreau M, Laporte G, Potvin J-Y. Metaheurística para o problema de roteamento de veículos. Relatório de pesquisa GERAD G-98-52, MontreTal, Canadá, 1998.

[6] Golden BL, Wasil EA, Kelly JP, Chao IM. O impacto da metaheurística na solução do problema de roteamento de veículos: algoritmos, conjuntos de problemas e resultados computacionais. Em: Crainic TG, Laporte G, editores. Fleet management and logistics. Dordrecht: Kluwer, 1998. p. 33-56.
[7] Van Breedam A. An analysis of the e$ect of local improvement operators in GA and SA for the vehicle routing problem. Documento de trabalho RUCA 96/14, Universidade de Antuérpia, Bélgica, 1996.
[8] Goldberg DE, Lingle R. Alleles, loci and the traveling salesman problem. Em: Grefenstette JJ. editor. Proceedings of the First International Conference on Genetic Algorithms, Hillsdale, NJ: Lawrence Erlbaum, 1985. p. 154 -9.
[9] Schmitt LJ. An empirical study computational study of genetic algorithms to solve order problems: an emphasis on TSP and VRPTC. Tese de doutorado, Universidade de Memphis, 1994.
[10] Schmitt LJ. An evaluation of a genetic algorithmic approach to the VRP (Avaliação de uma abordagem algorítmica genética para o VRP). Documento de trabalho, Department of Information Technology Management, Christian Brothers University, Memphis, 1995.
[11] Oliver IM, Smith DJ, Holland JRC. A study of permutation crossover operators on the traveling salesman problem (Um estudo de operadores de permutação cruzada no problema do caixeiro viajante). Em: Grefenstette JJ. editor. Proceedings of the Second International Conference on Genetic Algorithms [Anais da Segunda Conferência Internacional sobre Algoritmos Genéticos]. Hillsdale, NJ: Lawrence Erlbaum, 1987. p. 224 -30.
[12] Potvin J-Y. Genetic algorithms for the traveling salesman problem (Algoritmos genéticos para o problema do caixeiro viajante). Annals of Operations Research 1996;63:339-70.
[13] Potvin J-Y, Bengio S. O problema de roteamento de veículos com janelas de tempo - Parte II: pesquisa genética. INFORMS Journal on Computing 1996;8:165-72.
[14] Thangiah SR. Roteamento de veículos com janelas de tempo usando algoritmos genéticos. Relatório SRU-CpSc-TR-93-23, Slippery Rock University, Slippery Rock, 1993.
[15] Prins C. Competitive genetic algorithms for the open-shop scheduling problem. Mathematical Methods of Operations Research 2000;52(3):389-411.
[16] Lacomme P, Prins C, Ramdane-CheTrif W. A genetic algorithm for the Capacitated Arc Routing Problem and its extensions (Um algoritmo genético para o problema de roteamento de arcos capacitados e suas extensões). Em: Boers EJW et al., editores. Applications of evolutionary computing (Aplicações da computação evolutiva). Lecture Notes in Computer Science, vol. 2037. Berlim: Springer, 2001. p. 473-83.
[17] Beasley JE. Métodos Route-first cluster-second para roteamento de veículos. Omega 1983;11:403-8.
[18] Jaumard B, Semet F, Vovor T. A two-phase resource constrained shortest path algorithm for acyclic graphs. Relatório de pesquisa GERAD G-96-48, MontreTal, Canadá, 1996.
[19] Moscato P. Memetic algorithms: a short introduction. Em: Corne D, Dorigo M, Glover F, editores. New ideas in optimization (Novas ideias em otimização). Nova York: McGraw-Hill, 1999. p. 219-34.
[20] Clarke G, Wright JW. Programação de veículos de um depósito central para vários pontos de entrega. Operations Research 1964;12:568-81.
[21] Mole RH, Jameson SR. Um algoritmo de construção de rotas sequenciais que emprega um critério de economia generalizado. Operational Research Quaterly 1976;27:503-11.
[22] Gillett BE, Miller LR. A heuristic algorithm for the vehicle dispatch problem (Um algoritmo heurístico para o problema de despacho de veículos). Operations Research 1974;22:340-9.
[23] Cheung BKS, Langevin A, Villeneuve B. High performing evolutionary techniques for solving complex location problems in industrial system design. Journal of Intelligent Manufacturing 2001;12(5 - 6):455-66.
[24] The OR Library. http://mscmga.ms.ic.ac.uk/jeb/orlib.
[25] VRPs grandes. http://www-bus.colorado.edu/publications/workingpapers/kelly.
[26] Xu J, Kelly J. A network flow-based tabu search heuristic for the vehicle routing problem. Transportation Science 1996;30:379-93.
[27] Toth P, Vigo D. The granular tabu search and its application to the VRP. Relatório de pesquisa OR-98-9, DEIS, Universidade de Bolonha, 1998, a ser publicado no INFORMS Journal of Computing.
[28] Glover F, Laguna M, MartTi R. Scatter search. Em: Ghosh A, Tsutsui S, editores. Advances in evolutionary computing (Avanços na computação evolutiva):
teoria e aplicações. Berlim: Springer, 2002.

Christian Prins é professor de pesquisa operacional e engenharia industrial na Universidade de Tecnologia de Troyes (UTT), França. Ele atua como diretor do Departamento de Engenharia de Sistemas Industriais. Seus interesses de pesquisa são em logística, programação e aspectos de engenharia de software em otimização combinatória.