Igor Ribeiro Sucupira
igorrs@ime.usp.br

MÉTODOS HEURÍSTICOS GENÉRICOS: META-HEURÍSTICAS E HIPER-HEURÍSTICAS

Orientador:

Flávio Soares Corrêa da Silva
Departamento de Ciência da Computação
Instituto de Matemática e Estatística
fcs@ime.usp.br

Universidade de São Paulo
São Paulo - 2004

Resumo

As meta-heurísticas são estruturas algorítmicas gerais adaptáveis a diversos problemas de otimização. Muitos desses métodos têm sido amplamente estudados nas últimas décadas, resultando em algoritmos heurísticos de alta qualidade, que freqüentemente superam até mesmo heurísticas especializadas. Porém, uma mesma meta-heurística dificilmente é adequada a diversos tipos de problemas. Além disso, implementações realmente fortes de meta-heurísticas normalmente requerem conhecimentos específicos consistentes sobre o problema que se deseja tratar. Neste trabalho, faz-se uma ampla apresentação das meta-heurísticas existentes (incluindo-se uma recente aplicação concreta da meta-heurística *particle swarm optimization* - [27], [39], [40]). Concluindo, apresenta-se o conceito de hiper-heurística, proposto no ano 2000, que sugere uma estratégia para o desenvolvimento de heurísticas robustas. Idealmente, uma vez implementada uma hiper-heurística de qualidade, é possível adaptá-la de maneira simples para lidar desenvoltamente com diferentes problemas. Com três exemplos concretos recentes, ilustrar-se-á o potencial dos métodos hiper-heurísticos na resolução do conflito entre as heurísticas implementáveis em tempo reduzido com poucos conhecimentos específicos e as heurísticas de implementação simples.

1 Introdução ([29] e [34])

Um algoritmo é considerado um método heurístico quando não há conhecimentos matemáticos completos sobre seu comportamento, ou seja, quando, sem oferecer garantias, o algoritmo objetiva resolver problemas complexos utilizando uma quantidade não muito grande de recursos - especialmente no que diz respeito ao consumo de tempo - para encontrar soluções de boa qualidade.

A importância dos métodos heurísticos advém do bom desempenho médio - detectado de forma predominantemente experimental - desses métodos, quando aplicados ao tratamento de problemas (normalmente NP-difíceis) para os quais não são conhecidos métodos eficientes que forneçam garantias (como os algoritmos probabilísticos e de aproximação). Para um estudo aprofundado sobre problemas complexos, algoritmos de aproximação e algoritmos probabilísticos, [4] é uma referência recomendada.

As pesquisas - realizadas ao longo de décadas - sobre o desempenho de métodos heurísticos - e, em particular, sobre as características que conduzem ao êxito de tais métodos - levaram à elaboração de estratégias genéricas - esqueletos de algoritmos - para a construção de heurísticas. Essas estratégias são chamadas *meta-heurísticas*.

Embora não haja um consenso sobre a definição exata de meta-heurística, o conceito mais benquisto pode ser encontrado em [38]:

"Uma meta-heurística é um conjunto de conceitos que pode ser utilizado para definir métodos heurísticos aplicáveis a um extenso conjunto de diferentes problemas. Em outras palavras, uma meta-heurística pode ser vista como uma estrutura algorítmica geral que pode ser aplicada a diferentes problemas de otimização com relativamente poucas modificações que possam adaptá-la a um problema específico. Alguns exemplos de meta-heurísticas são: *simulated annealing*, busca tabu, *iterated local search*, algoritmos evolutivos e *ant colony optimization*".

As meta-heurísticas dificilmente alcançam o mesmo desempenho de métodos heurísticos especializados. De fato, a importância das meta-heurísticas deve-se a outras características. Por descreverem métodos adaptáveis, as meta-heurísticas fornecem idéias que podem ser aplicadas aos problemas de otimização para os quais não são conhecidos algoritmos - nem mesmo heurísticos - específicos eficientes. Além disso, esses métodos têm sido amplamente estudados nas últimas décadas, o que aprofundou de maneira marcante o conhecimento sobre o processo de resolução de problemas complexos.

Contudo, o impacto comercial das meta-heurísticas tem sido bem menor que o esperado, pois a adaptação de uma meta-heurística para o tratamento eficiente e eficaz de um novo problema é, em muitos casos, uma tarefa custosa - dependente de fortes conhecimentos sobre o problema. Devido a esses obstáculos, usuários com pequena disponibilidade de recursos continuam se utilizando de métodos clássicos fracos - mas de implementação barata -, como os algoritmos gulosos.

A popularidade insatisfatória das meta-heurísticas no meio comercial levou o foco de muitos pesquisadores à necessidade de se desenvolverem métodos heurísticos que possam operar com desenvoltura em um nível maior de generalidade. Dessa necessidade surgiu o conceito de *hiper-heurística*, descrito a seguir.

Uma hiper-heurística tem a função de viabilizar a utilização inteligente de diversos métodos heurísticos no mesmo processo de resolução de um problema. Mais especificamente, uma hiper-heurística é uma heurística que é ativada a cada ponto de decisão de um algoritmo, determinando a melhor heurística a ser utilizada no próximo passo da resolução. Na raiz dessa idéia formalizada recentemente e ainda pouco estudada, está a intenção de que um algoritmo bem definido seja capaz de, utilizando-se de um conjunto mutável de heurísticas razoavelmente simples, obter sucesso no tratamento de diversos problemas, eliminando o conflito entre as estratégias fortes e as estratégias baratas.

2 Meta-heurísticas

Esta seção se pretende uma introdução razoavelmente ampla às meta-heurísticas. Nela, apresenta-se uma visão geral sobre os diversos tipos de meta-heurísticas, bem como a descrição de um exemplo concreto de aplicação.

2.1 Visão geral ([29])

Embora todos os procedimentos descritos por meta-heurísticas possam ser vistos como estratégias de busca, algumas características desses procedimentos permitem classificar a maioria das meta-heurísticas em tipos bastante distintos, de acordo com a abordagem utilizada: evolução de populações compostas por soluções, construção gradual inteligente, busca por entornos e relaxação.

Meta-heurísticas construtivas

Uma meta-heurística construtiva estabelece estratégias para a construção de uma solução através da definição, de forma meticulosa, do valor de cada uma de suas componentes. Dentre essas meta-heurísticas, a mais clássica é a estratégia gulosa, formada por um processo iterativo que, em cada passo, adiciona um elemento à solução parcial, de forma a obter os melhores resultados imediatos possíveis. Um exemplo híbrido mais eficaz é a meta-heurística *GRASP* (*Greedy Randomized Adaptive Search Procedure* - [33]), formada pela repetição de um algoritmo de duas fases: construção gradual inteligente e busca por entornos.

Na meta-heurística GRASP, a fase construtiva é um processo iterativo que parte de uma solução parcial vazia e, em cada passo: identifica todos os elementos que podem ser incorporados à solução parcial sem torná-la inviável; cria um subconjunto desses elementos, contendo apenas aqueles que causam aumento mínimo de custo (este é o aspecto guloso da meta-heurística); seleciona aleatoriamente um elemento desse subconjunto, acrescentando-o à solução parcial.

Meta-heurísticas de relaxação

As meta-heurísticas de relaxação realizam alterações na modelagem original do problema a ser resolvido, construindo, desta maneira, um problema mais simples - *problema*

relaxado -, cuja solução pode ser encontrada com eficiência, fornecendo informações que guiarão o algoritmo na busca da solução do problema original.

A qualidade de uma meta-heurística de relaxação, portanto, está relacionada quase exclusivamente à eficiência do processo de resolução do problema relaxado e à utilidade da solução obtida - no sentido de tornar mais eficiente e eficaz o processo de resolução do problema original.

As características comumente alteradas para a criação de um problema relaxado são o conjunto de restrições e a função objetivo. Alterar o conjunto de restrições - removendo ou enfraquecendo restrições -, pode criar maior liberdade de movimentos no espaço de busca, tornando a solução ótima alcançável também a partir de regiões inviáveis. Modificar a função objetivo é especialmente importante no tratamento de problemas em que o cálculo do custo exato das soluções tem impacto excessivo no consumo de tempo do algoritmo.

Um exemplo simples de relaxação é a eliminação, em um problema de programação linear inteira, da restrição de que as variáveis devem ser inteiras. Desta maneira, o problema relaxado pode ser resolvido com a utilização de um algoritmo eficiente - como o método Simplex - e sua solução pode ser proposta - após modificações simples que tornem suas variáveis inteiras - como uma solução do problema original.

Os métodos de *relaxação lagrangeana* ([20]) constituem um exemplo mais geral de meta-heurística de relaxação. Sejam: λ um vetor de números não negativos - chamados *multiplicadores de Lagrange* - e P um problema de minimização - com função objetivo f - que possui, entre suas restrições, um conjunto (na forma $Ax \leq b$) de inequações sem as quais a resolução de P seria simples. Neste caso, a relaxação lagrangeana de P é um problema LR que possui o mesmo conjunto de restrições de P, exceto por $Ax \leq b$. A função objetivo de LR é g tal que $g(x) = f(x) + \lambda(Ax - b)$. Se o conjunto de restrições a serem removidas de P possui a forma $Ax = b$, a função g será definida como $g(x) = f(x) + \lambda_1(Ax - b) + \lambda_2(-Ax + b)$, sendo λ_1 e λ_2 vetores de multiplicadores de Lagrange.

Meta-heurísticas de busca por entornos

A *vizinhança* de uma solução S é o conjunto de soluções que podem ser geradas aplicando-se algum operador a S. Assim, as meta-heurísticas de busca por entornos

podem ser definidas como procedimentos que percorrem espaços de busca (compostos por soluções) levando fundamentalmente em conta, em cada passo, a vizinhança da solução em mãos.

O conjunto de operadores aplicáveis para a construção de uma vizinhança depende não somente do problema como também do algoritmo, mas é de extrema importância para o êxito de um processo de busca por entornos. Uma estrutura de entornos muito grande, por exemplo, tende a diminuir o desempenho de cada iteração, enquanto uma estrutura de entornos com baixa diversificação pode limitar o espaço de busca de forma decisivamente negativa.

Outros aspectos importantes - porém dependentes do problema - normalmente não abordados na definição de uma meta-heurística são o cômputo da função objetivo, a geração da solução inicial e o *critério de parada* - que indica o que deve ser avaliado para que se possa decidir, a cada iteração, se a busca deve ou não continuar. A primeira dificuldade é comumente tratada com técnicas de relaxação ou com estratégias que permitam calcular o custo da solução levando em conta apenas a solução anterior - incluindo seu custo - e o operador que foi utilizado entre ambas. Para a geração da solução inicial, a estratégia mais comum é a utilização de processos construtivos (veja, acima, a meta-heurística GRASP). O critério de parada pode ser definido de diversas maneiras, levando-se em conta aspectos como o tempo total de execução, o tempo sem que se tenham obtido progressos ou o número de movimentos efetuados no espaço de busca. O tempo pode ser considerado de diversas maneiras - por exemplo, em iterações ou em segundos de CPU.

Em sua concepção primária, as buscas por entornos eram essencialmente monótonas, o que pode ser sintetizado como a meta-heurística *hill-climbing*. Na forma básica, essa meta-heurística parte de uma solução inicial e, em cada iteração, realiza um movimento que conduz a solução atual (S) a uma solução, de sua vizinhança, melhor que S. Quando não existe uma solução melhor na vizinhança da solução atual (S_f), diz-se que foi encontrado um *ótimo local* - S_f. Uma variação da meta-heurística *hill-climbing* é sua versão gulosa, que, em cada iteração, parte da solução atual para a melhor solução de sua vizinhança. Outra variação - mais eficiente que a estratégia gulosa - é a meta-heurística de

intensificação oscilante dinâmica, que altera dinamicamente a intensidade[1] da busca, de forma a aumentar a eficiência do processo nas fases em que o entorno contém muitas possibilidades de diminuição de custo e a torná-lo mais minucioso quando há escassez de soluções com essas características - o resultado prático tende a ser uma busca que tem baixa intensidade nas primeiras fases e torna-se exaustiva nas proximidades do ótimo local.

Em geral, as meta-heurísticas de busca monótona têm como grande inconveniente o fato de que, dependendo da solução inicial escolhida e das características da instância do problema, elas freqüentemente ficam limitadas a uma pequena região do espaço de busca, cujo ótimo local pode ter custo muito maior que o da solução ótima. Para a realização de buscas que procuram evitar essa limitação, as principais técnicas conhecidas baseiam-se em pelo menos uma das seguintes estratégias: reinício, alteração da estrutura de entornos - ou da função objetivo - e não-monotonicidade.

As meta-heurísticas de *partida múltipla* realizam diversas buscas monótonas, com diferentes soluções iniciais aleatórias, de forma a explorar regiões variadas do espaço de busca, aumentando a qualidade da melhor solução encontrada, bem como a probabilidade de que esta seja ótima.

A meta-heurística *VNS* (*Variable Neighbourhood Search* - [21]) realiza uma busca monótona alterando sistematicamente a estrutura de entornos, com o apoio de um conjunto *C* de estruturas de entornos definidas. Mesmo em suas formas mais simples, esta meta-heurística possui diversas variações. A busca *VND* (*Variable Neighbourhood Descent*), por exemplo, encontra o melhor ponto da vizinhança da solução em mãos e passa à próxima estrutura de entornos de *C* - se não houver aumento de qualidade - ou retorna à primeira delas - em caso contrário. Já a busca *RVNS* (*Reduced VNS*) substitui a escolha do melhor ponto da vizinhança pela escolha de um ponto aleatório - aumentando a eficiência do processo -, enquanto as buscas *BVNS* (*Basic VNS*) e *GVNS* (*General VNS*) combinam características das duas anteriores.

A meta-heurística de *Busca Local Guiada* (*Guided Local Search* - [36]) é uma extensão dos métodos de *hill-climbing*, apresentando qualidade muito superior à dessas estratégias

1 A intensidade de uma busca por entornos é a quantidade de soluções analisadas na vizinhança da solução em mãos.

simples. A idéia básica da Busca Local Guiada é penalizar - através de alterações na função objetivo - os elementos que aparecem com freqüência em ótimos locais, de forma a incentivar a exploração de diversas regiões do espaço de busca. Numericamente falando, deve-se atribuir um custo - normalmente estático e baseado na função objetivo - e um fator de penalização - dinâmico e inicializado com zero - a cada possível componente de soluções. Sempre que a busca encontra um ótimo local, selecionam-se alguns componentes desta solução - com base nos fatores de penalização e nos custos - e incrementam-se os fatores de penalização destes componentes. Com isso, a nova função objetivo indicará maior custo às soluções que contenham os componentes selecionados.

A meta-heurística *Recozimento Simulado* (*Simulated Annealing* - [3]) é um método bastante clássico que prescinde da monotonicidade, realizando cada uma de suas iterações da seguinte maneira: sorteia-se uma solução (S_S, com custo $f(S_S)$) da vizinhança da solução corrente (S_C, com custo $f(S_C)$) e atualiza-se a solução corrente com probabilidade $e^{\Delta E/T}$, onde $\Delta E = min\{0, f(S_C)-f(S_S)\}$ e T é um número positivo dinamicamente ajustável. Desta maneira, os movimentos que promovem aumento de qualidade são sempre permitidos, enquanto os demais ocorrem com uma probabilidade que decresce exponencialmente com o aumento da diferença de custo. Se forem atribuídos valores próximos de zero ao parâmetro T, a busca se comportará de forma semelhante a um algoritmo de *hill-climbing*.

A meta-heurística *Threshold Accepting* ([10]) - uma simplificação do Recozimento Simulado - atualiza a solução corrente se, e somente se, $f(S_S)-f(S_C) \leq t$, onde t é um limitante dinamicamente ajustável.

A *Busca Tabu* ([1], [18]) é uma meta-heurística de busca não-monotônica cuja principal característica é a capacidade de exploração do histórico do processo de busca, organizado em estruturas que compõem o que se chama de *memória adaptativa*. A construção e a exploração da memória adaptativa levam em conta quatro dimensões, relacionadas à freqüência, à presença no passado recente, à qualidade e à influência. Por exemplo, podem ser mantidas tabelas com: os componentes que aparecem com maior freqüência, os componentes mais freqüentes em soluções críticas[2] e os componentes removidos ou

2 O conceito de solução crítica pode ser definido, por exemplo, da seguinte maneira: S_i é uma solução crítica se, e somente se, possui qualidade superior à das soluções S_{i-1} e S_{i+1} (para todo n, S_n é a n-ésima solução encontrada no processo de busca).

adicionados à solução no passado recente. Este último tipo de tabela - chamado *lista tabu* - é comumente utilizado para classificar elementos como *tabu*, não permitindo, durante um certo número de iterações, que esses sejam removidos da (ou adicionados à) solução. Um dos métodos mais comuns, no que diz respeito à qualidade, são os *critérios de aspiração*, que podem tornar possível a aceitação de uma solução de alta qualidade, desconsiderando elementos de impedimento, como, por exemplo, componentes tabu.

Dois conceitos aos quais se fazem muitas referências no contexto da Busca Tabu - e de grande influência também na qualidade de outras meta-heurísticas - são a *intensificação* (exploração do espaço de busca através de componentes e de tipos de movimentos que, historicamente, levam a soluções de boa qualidade) e a *diversificação* (tentativa de construção de soluções que pertencem a regiões não exploradas do espaço de busca e diferem significativamente das soluções já encontradas).

A *Busca Reativa* (*Reactive Search* - [6], [7]) propõe uma extensão da Busca Tabu, especialmente no que diz respeito à diversificação. Utilizam-se estruturas de dados especiais, para explorar a memória em longo prazo através da verificação explícita da existência de ciclos. Estes são combatidos com elevações no tamanho da lista tabu ou, em casos extremos, com a realização de uma seqüência de movimentos aleatórios.

Meta-heurísticas evolutivas

As meta-heurísticas evolutivas lidam com uma população - essencialmente, um conjunto - de soluções (possivelmente inviáveis) que evolui através da interação entre seus elementos, procurando preservar as características desejáveis (de forma a melhorar, ao longo das gerações, a qualidade média das soluções), sem comprometer a diversidade dentro da população. Alguns exemplos de meta-heurísticas evolutivas são os *algoritmos genéticos* ([23], [32]), os *algoritmos meméticos* ([30]), os *algoritmos de estimação de distribuição* ([26]), a *busca dispersa* (*scatter search* - [24], [25]) e *path relinking* ([24], [25]).

Nos algoritmos genéticos, cada elemento da população é chamado *cromossomo*, sendo a codificação de uma solução na forma de uma seqüência de símbolos - chamados *genes*. Um algoritmo genético também deve definir uma função que mede a aptidão de cada indivíduo (naturalmente, indivíduos que representam soluções de baixo custo tendem a

ser mais aptos, enquanto os indivíduos que representam soluções inviáveis têm baixa aptidão). A população é alterada através de dois operadores principais: a mutação - que modifica um dos genes de um indivíduo e tem baixa probabilidade de ocorrer - e a recombinação - que constrói um novo indivíduo utilizando apenas a informação genética de dois outros indivíduos selecionados aleatoriamente. Indivíduos com baixa aptidão têm menor probabilidade de serem selecionados para recombinação.

Os algoritmos meméticos são derivados dos algoritmos genéticos e diferem desses por incorporarem, além dos operadores de mutação e recombinação, processos de busca por entornos de soluções individuais. Em um algoritmo memético, cada elemento da população é chamado de *agente* e pode aplicar meta-operadores (chamados otimizadores locais), que alteram repetidamente (através do operador de mutação) a solução contida no agente, visando a um aumento específico da aptidão desse agente.

Um algoritmo de estimação de distribuição constrói aleatoriamente uma população inicial de soluções e, em cada iteração, realiza os seguintes passos: seleciona os elementos mais aptos da população, que formarão um conjunto *S*; estima a distribuição de probabilidade conjunta dos indivíduos selecionados, ou seja, a freqüência com que cada característica possível das soluções aparece em *S*; substitui a população atual por uma nova, criada aleatoriamente com base na distribuição de probabilidade estimada.

A busca dispersa difere das meta-heurísticas evolutivas apresentadas até aqui por empregar técnicas sistemáticas (e não aleatórias) na substituição de uma população por outra. Nesse sentido, a busca dispersa promove a intensificação e a diversificação utilizando mecanismos que a assemelham à busca tabu. Mais especificamente, o algoritmo constrói um conjunto inicial de soluções e aplica métodos heurísticos para melhorar a qualidade de cada uma dessas soluções. Em seguida, um subconjunto das melhores soluções é selecionado para ser o *conjunto de referência*. Após essas fases iniciais, é realizado um processo iterativo que, em cada passo: cria novas soluções a partir de combinações (convexas e não convexas) de subconjuntos do conjunto de referência atual; aplica métodos heurísticos para melhorar a qualidade individual das soluções criadas; substitui o conjunto de referência atual por um subconjunto das melhores soluções existentes. A qualidade das soluções é medida de acordo com critérios diversos, levando em conta, por exemplo, a função objetivo e a diversidade.

A meta-heurística *path relinking* assemelha-se muito à busca dispersa e pode ser considerada uma extensão desta, especialmente no que diz respeito ao mecanismo de combinação de soluções: para a construção de combinações entre duas soluções, parte-se de uma delas e gera-se um caminho de soluções até a outra (chamada *solução-guia*). Obviamente, também se podem inverter os papéis entre as duas soluções, ou mesmo construir-se o caminho fazendo-se com que cada uma das soluções se movimente em direção à outra. Os operadores aplicados realizam remoções, adições ou modificações de elementos da solução.

Outros tipos de meta-heurísticas

Muitas meta-heurísticas importantes não se enquadram fortemente em nenhuma das categorias acima, por estarem focadas na utilização de algum recurso computacional especial ou por incorporarem intensamente elementos de mais de um tipo de meta-heurística.

As *meta-heurísticas de decomposição* apresentam características intermediárias entre os métodos construtivos e os de relaxação, pois estabelecem estratégias para a divisão de uma instância de problema em instâncias menores e para a integração das soluções desses subproblemas na construção de uma solução para a instância original. Este tipo de meta-heurística é extremamente útil para a implementação de algoritmos que envolvem paralelização.

As *redes neurais artificiais* ([3]) são modelos abstratos - baseados nos sistemas nervosos naturais - formados por unidades interconectadas, chamadas *neurônios*. Uma rede neural tem a capacidade de, dado um conjunto de dados de entrada com os correspondentes dados de saída, ajustar seus parâmetros para tornar-se capaz de aproximar o valor de uma determinada função. Em um problema de otimização, esta função tem como domínio o conjunto das possíveis instâncias do problema e como imagem o conjunto das soluções (no sentido original, ligado à otimalidade) dessas instâncias. As principais vantagens das redes neurais artificiais são a adaptação eficiente a alterações (de modelo ou de instância), a estrutura facilmente paralelizável e a possibilidade de utilização de hardware específico.

A meta-heurística *Ant Colony Optimization* (*ACO* - [16]) baseia-se no comportamento utilizado pelas colônias de formigas para traçar rotas entre o formigueiro e as fontes de

alimentação. O principal aspecto desse comportamento é uma substância - chamada feromônio - que é secretada pelas formigas durante seus percursos, de forma a indicar caminhos prometedores a outras formigas. A idéia básica da meta-heurística é utilizar formigas artificiais, representadas por processos concorrentes, que traçam caminhos em um grafo cujos vértices representam componentes da solução.

A meta-heurística de *Otimização Extrema* (*Extreme Optimization* - [9]) baseia-se em ecossistemas que evoluem através da extinção de indivíduos mal-adaptados, ou seja, nos processos evolutivos cujo foco está nos elementos indesejáveis. Contudo, a Otimização Extrema não é considerada uma meta-heurística evolutiva, pois não é baseada em populações de soluções. Após a construção de uma solução inicial, executa-se repetidamente um processo que, com base em uma função de aptidão, seleciona o pior componente da solução e o remove, realizando as alterações necessárias para que a solução continue consistente, sem destruição significativa dos ganhos obtidos em passos anteriores.

A meta-heurística *Particle Swarm Optimization* ([39], [40]) baseia-se no comportamento social dos pássaros em revoadas. Algoritmicamente, tem-se um conjunto de partículas que percorrem o espaço de busca apresentando comportamentos aleatórios em relação à *individualidade* e à *sociabilidade*[3]. A individualidade de uma partícula está relacionada à ênfase dada, em seus movimentos, à melhor solução já encontrada por ela mesma, enquanto sua sociabilidade reflete o grau de importância dado por ela à melhor solução já encontrada por seus vizinhos. O conceito de vizinhança em PSO não é o mesmo utilizado pelas meta-heurísticas de busca por entornos, pois cada partícula, associada a uma solução que evolui, é vizinha de um conjunto de partículas que nunca é alterado. Além disso, é importante notar que a estrutura de vizinhanças é construída de forma que os progressos obtidos em cada região tenham influência, potencialmente, em todas as partículas.

A meta-heurística *Iterated Local Search* (*ILS* - [28]) consiste na aplicação iterativa de uma heurística - *H* - cujo funcionamento não é necessariamente conhecido pela estrutura principal do algoritmo[4]. Mais especificamente, cada passo de um algoritmo ILS deve:

3 Observe, adiante, que esses conceitos não são complementares.

4 Observe que, mesmo que *H* seja um algoritmo de *hill-climbing*, a meta-heurística ILS não é um processo de busca *hill-climbing* com reinício aleatório, pois a solução utilizada como ponto de partida, em cada passo, é construída com base no histórico do processo.

efetuar mutações na solução corrente, com base nas soluções já visitadas nas iterações anteriores, de forma a gerar uma nova solução s_1; aplicar H à solução s_1, obtendo um ótimo local s_2; fazer de s_2 a solução corrente, caso ela satisfaça um determinado critério de aceitação baseado na solução corrente, na solução s_1 e no histórico da busca. A idéia da meta-heurística ILS é, considerando H como uma função de imagem S^* (o conjunto de ótimos locais), efetuar, idealmente, uma busca por entornos em S^*.

A meta-heurística *Fuzzy Adaptive Neighbourhood Search* (*FANS* - [8]), bastante genérica, utiliza conceitos baseados na teoria de conjuntos *Fuzzy* ([5]) para propor uma estrutura algorítmica adaptável em diversos aspectos, notadamente quanto ao critério de aceitação de novas soluções (o que faz com que, por exemplo, *Simulated Annealing* seja um caso particular de FANS) e à seleção dos operadores utilizados em cada iteração (o que assemelha estes métodos às meta-heurísticas de busca por entornos variáveis).

2.2 Um exemplo concreto: minimização da largura de banda

Em uma recente aplicação concreta da meta-heurística *Particle Swarm Optimization* (PSO), Lim et al. ([27]) propuseram métodos para a resolução do problema da minimização da largura de banda (*bandwidth minimization problem* - BMP), que, dado um grafo $G = (V, E)$, consiste na determinação de uma permutação p dos vértices de G, de forma a minimizar o valor de $max\{|p(i) - p(j)|: (i, j) \text{ é uma aresta de } E\}$. Este problema NP-completo, muito conhecido, possui diversas aplicações: na multiplicação de matrizes esparsas, no projeto de circuitos, em geofísica numérica etc.

O algoritmo proposto em [27] - uma versão com otimização local (*hill-climbing*) da meta-heurística PSO - consiste na repetição, por um determinado número de vezes, do seguinte processo: para cada agente i, encontre um ótimo local L_i partindo da solução atualmente em i; atualize o apontador para o melhor agente, se necessário; ajuste a velocidade de cada agente; "mova" o agente, com a velocidade calculada no passo anterior. Tanto o número de iterações quanto o número de agentes são ajustados de acordo com o tamanho do grafo.

Sejam:

- $n = |V|$.

- $X_i = (x_{i1}, x_{i2}, ..., x_{in})$ a posição atual do agente i (mais especificamente, uma permutação dos vértices de G). Por exemplo: $x_{i2} = 3$ indica que o vértice 2 está rotulado com o número 3 na posição corrente do agente i.
- S_i a velocidade do agente i.
- $P_i = (p_{i1}, p_{i2}, ..., p_{in})$ a melhor posição em que o agente i já esteve (ou seja: a melhor solução já encontrada pelo agente i).
- $P_g = (p_{g1}, p_{g2}, ..., p_{gn})$ a melhor solução já encontrada em todo o processo de busca.

Pelas especificações da meta-heurística PSO, o cálculo da nova velocidade de cada agente i, bem como a movimentação do agente, em cada passo, seriam dados, respectivamente, pelas fórmulas:

- Para todo d, $s_{id} = \omega \cdot s_{id} + \eta_1 \cdot rand() \cdot (p_{id} - x_{id}) + \eta_2 \cdot rand() \cdot (p_{gd} - x_{id})$, onde ω, η_1 e η_2 são parâmetros fixos.
- Para todo d, $x_{id} = x_{id} + s_{id}$.

No algoritmo implementado por Lim et al., o valor P_i, para cada agente i, foi substituído pelo valor de L_i, ou seja, utiliza-se o atual ótimo local do agente, em detrimento de sua experiência, o que acelera a convergência do processo. Além disso, observe que as fórmulas, apesar de facilmente adaptáveis a problemas com domínios contínuos, requerem ajustes para o tratamento do problema BMP. Para realizar tais ajustes, os autores redefiniram o conceito de velocidade, bem como as operações de soma (entre posição e velocidade), de subtração (entre posições) e de multiplicação (entre coeficiente e velocidade).

A velocidade S de um agente é definida como uma lista de $|S| = t$ transposições: $\{(i_1, j_1), (i_2, j_2), ..., (i_t, j_t)\}$. Tal velocidade indica que, partindo-se da transposição (i_1, j_1), deve-se, para cada transposição (i_k, j_k), permutar os rótulos dos vértices i_k e j_k. Para se obter a posição resultante da aplicação de uma velocidade S a uma posição P, realizam-se as transposições de S, na ordem dada, em P. A velocidade resultante da multiplicação de uma velocidade S por um coeficiente racional c é simplesmente uma seqüência de $c \cdot |S|$ transposições, partindo-se da primeira transposição de S.

A operação mais complexa definida por Lim et al. é a subtração entre posições. Para se

definir tal operação, considera-se, para cada posição P e cada rótulo k, o valor *inversa_P [k]*, que indica a qual vértice está associado o rótulo k na permutação P. Com isso, dadas permutações P_1 e P_2, calcula-se a velocidade S resultante da operação P_1 - P_2 através de um processo iterativo que se inicia com $S = \{\}$. Partindo-se do vértice 1, compara-se, para cada vértice k, os valores de $P_1[k]$ e $P_2[k]$. Quando há diferença, concatena-se a S a transposição $(k, inversa\_P_1[P_2[k]])$. Ao fim do processo, a velocidade S será tal que $P_2 + S = P_1$.

Para a geração da população inicial do algoritmo, Lim et al. particionam, diversas vezes, os vértices em níveis N_0, ..., N_m, de forma que, para toda aresta (w, u), os vértices w e u pertençam ao mesmo nível ou a níveis consecutivos. Realizando-se este particionamento através de uma busca em largura e rotulando-se os vértices um nível por vez (a partir do rótulo 1 e do nível N_0), a largura de banda resultante será um valor b tal que $T \leq b \leq 2T-1$, sendo T o número de vértices do nível que contém mais vértices.

O algoritmo descrito acima para o tratamento do problema BMP foi implementado em duas versões. A primeira versão monitora o movimento de cada agente e, ao detectar que um determinado agente a não obtém progresso há diversos movimentos, substitui a por um novo agente, também gerado com a realização de uma busca em largura. A segunda versão do algoritmo implementada difere da anterior apenas por não possuir o mecanismo de detecção de progresso.

O algoritmo de *hill-climbing* utilizado em ambas as versões do algoritmo de Lim et al. recebe uma permutação p e se utiliza, em cada iteração, das seguintes definições:

- Para todo vértice v, $B(v) = max\{|p(u) - p(v)|: u \text{ é vizinho de } v\}$.
- Um vértice v é *crítico* se $B(v)$ é igual à largura de banda da permutação p.
- Para cada vértice crítico v, $max(v)$ é o rótulo de maior valor dentre os vértices vizinhos de v. O valor $min(v)$ é análogo.
- Para cada vértice crítico v, $mid(v) = (max(v) + min(v))/2$ (observe que este é o rótulo ideal para o vértice v, na permutação p).
- Para cada vértice crítico v e todo vértice w, $dist(v, w) = |mid(v) - p(w)|$.
- Para cada vértice crítico v, $W(v) = \{w: dist(v, w) < dist(v, v)\}$.

Cada iteração do algoritmo de *hill-climbing* identifica o conjunto *C* de vértices críticos e, para cada vértice *v* de *C*, realiza o seguinte processo: considerando os vértices de *W(v)* em ordem não decrescente de *dist(v, w)*, encontre um vértice *w* tal que, ao se permutarem os rótulos de *v* e *w*, o valor de *B(v)* seja reduzido e *B(w)* não se eleve - a não ser que *w* não esteja em *C*, caso em que apenas se deve verificar que *w* não passou a ser um vértice crítico.

Os experimentos realizados por Lim et al. contemplam as duas versões, implementadas em Java, da meta-heurística PSO, procurando compará-las com as quatro melhores heurísticas (nomeadas, por Lim et al., como GPS, WBRA, TS e GRASP_PR) conhecidas, à época, para o problema BMP. Os valores dos parâmetros das equações foram definidos como $\omega = 0.2$, $\eta_1 = 0.5$ e $\eta_2 = 3$ e a taxa de substituição de agentes (número de iterações suficientes para que, na ausência de progresso, um agente seja substituído) como *n/60*. Todas as instâncias do problema utilizadas nos experimentos foram obtidas da coleção Harwell-Boeing ([37]) de matrizes esparsas.

Na primeira bateria de testes, 14 casos de instâncias reais, com *n* variando entre 59 e 1005, foram utilizados para a comparação entre a qualidade das soluções encontradas pelos algoritmos GPS, WBRA, TS e PSO. O algoritmo de Lim et al. (em sua versão mais eficaz, com substituição de agentes) superou todos os outros métodos, na grande maioria dos casos. Na segunda fase de testes, utilizaram-se 33 instâncias, com *n* entre 30 e 199, enquanto a terceira fase se utiliza de 80 instâncias, com tamanhos entre 200 e 1000. Estas duas últimas baterias de testes, que comparam os métodos GPS, TS, GRASP_PR e PSO, apontam novamente para a superioridade do algoritmo PSO com substituição de agentes. Porém, o algoritmo PSO sem substituição de agentes e a heurística GRASP_PR produzem resultados semelhantes com maior velocidade. Esta última versão do algoritmo PSO apresentou-se duas (bateria 2) ou três (última bateria) vezes mais rápida que o algoritmo GRASP_PR, que, por sua vez, executou com o dobro da eficiência da primeira versão do algoritmo PSO.

3 Hiper-heurísticas

Esta introdução às hiper-heurísticas - fundamentalmente baseada em [34] - inicia-se com um breve histórico sobre os métodos para utilização dinâmica de heurísticas, considerados como precursores do conceito mais genérico e recente aqui abordado. Em seguida, apresentam-se algumas motivações para o estudo e a aplicação de hiper-heurísticas, bem como algumas definições mais claras sobre o assunto. A arquitetura sintetizada por Ross et al. ([34]), além de alguns algoritmos e sistemas surgidos nos últimos anos, fecham esta seção.

3.1 Evolução

Idéias precursoras em relação à utilização dinâmica de heurísticas podem ser encontradas, já na década de 80, inseridas na área de planejamento automático. Nesse sentido, talvez o exemplo mais notável seja o sistema PRODIGY ([2]), que, através de aprendizagem automatizada, evoluía seus processos de tomada de decisões. Porém, os primeiros casos conhecidos de seleção explícita de heurísticas em pontos de decisão múltiplos só vieram à tona na década de 90. Desses, pode-se destacar o agendador LR-26 (parte do sistema COMPOSER - [19]), utilizado no planejamento de comunicações entre satélites artificiais e três estações em terra (um problema de programação inteira binária, com centenas de variáveis e milhares de restrições).

O algoritmo utilizado pelo LR-26 dividia-se em 5 fases: definir multiplicadores de lagrange, para a criação de um problema relaxado; encontrar uma solução que não necessariamente satisfaça a todas as restrições; ordenar o conjunto de restrições não satisfeitas; propor algumas valorações, visando à viabilidade da solução; percorrer as valorações propostas. Para a efetiva implementação desse processo, havia 28 heurísticas disponíveis para as diferentes fases (o que representava 2592 estratégias diferentes), fazendo necessária uma avaliação experimental das estratégias. Devido à inviabilidade do processo de avaliação de todas as estratégias com os 50 casos de teste disponíveis, foi realizada uma seleção preliminar heurística, que reduziu para 51 o número de estratégias disponíveis, viabilizando o teste exaustivo de cada uma delas. O resultado da seleção foi um ganho decisivo de eficiência e eficácia, apesar da dependência em relação à representatividade dos casos de teste.

Em 1994, Fang et al. ([17]) propuseram um método (à época chamado de *evolving heuristic choice* - que tem como possível tradução a expressão "evolução das seleções de heurísticas") para a utilização de um algoritmo genético na seleção das heurísticas a serem aplicadas na resolução do problema de agendamento conhecido como *open-shop scheduling*. Cada instância desse problema consiste em alguns serviços, compostos por tarefas que podem ser agendadas em qualquer ordem. Cada tarefa deve executar em uma máquina específica, durante um tempo fixo. Sejam *m* o número de serviços e *n* o número de tarefas. No algoritmo genético desenvolvido, cada cromossomo possuía *n-1* pares de genes, de forma que cada *i*-ésimo par (s_i, h_i) indicava o passo a ser realizado na *i*-ésima iteração: selecionar o s_i-ésimo serviço incompleto (considerando a lista de serviços incompletos como circular) e escolher, utilizando a heurística h_i, uma de suas tarefas para ser colocada na primeira posição possível da agenda. Alguns exemplos de heurísticas utilizadas eram: selecionar a tarefa com menor tempo de processamento; selecionar a tarefa com maior tempo de processamento; dentre as tarefas que podem ser agendadas na primeira posição possível, selecionar a que tenha maior tempo de processamento.

Na estratégia proposta por Fang et al., há duas desvantagens principais: o tamanho dos cromossomos em instâncias de grande porte e a especificidade do algoritmo. Porém, esse método já pode ser considerado uma hiper-heurística - termo cunhado apenas no ano 2000 -, como ficará claro a seguir.

3.2 O conceito de hiper-heurística

O termo *hiper-heurística* se refere aos processos algorítmicos que se utilizam de heurísticas em um alto nível de generalidade, uma vez que o domínio de uma hiper-heurística é composto não por instâncias para o problema em mãos mas por conjuntos de heurísticas que podem ser utilizadas na busca por soluções de um problema que, possivelmente, não é conhecido pela hiper-heurística. A implementação mais óbvia de uma hiper-heurística consiste em um processo que, em cada ponto de decisão, seleciona uma heurística - idealmente, a mais adequada - para ser utilizada no próximo passo do algoritmo. Exemplos desse tipo de hiper-heurística são os algoritmos construtivos que, em cada iteração, selecionam a heurística que será responsável por acrescentar um elemento à solução parcial.

A principal motivação para o estudo de hiper-heurísticas é a combinação entre robustez e simplicidade. Potencialmente, uma hiper-heurística é um algoritmo bem definido, utilizável com eficiência e eficácia no tratamento de diversos problemas de otimização, exigindo, para isso, apenas a alteração do conjunto de heurísticas simples que lidam diretamente com o problema. Em outras palavras, o ideal que as abordagens hiper-heurísticas procuram alcançar é o da insensibilidade às falhas apresentadas pelas heurísticas pouco elaboradas. Os erros de decisão praticados por cada um desses métodos seriam suprimidos em um processo de intercalação entre heurísticas.

Naturalmente, a simplicidade não é característica obrigatória das hiper-heurísticas. Aplicações desenvolvidas com maior quantidade de recursos podem incorporar heurísticas mais complexas (baseadas em meta-heurísticas de qualidade, por exemplo) ao conjunto de métodos que operam diretamente sobre o problema. Além disso, nada impede que conhecimentos específicos sejam utilizados na própria hiper-heurística. Recentemente, Burke et al. ([11]) investigaram a utilização de raciocínio baseado em casos (*case based reasoning*) para a seleção de heurísticas em problemas de agendamento de cursos. O sistema mantém uma base de informações sobre o desempenho de cada heurística em instâncias previamente resolvidas. Para o treinamento do sistema, utilizam-se técnicas de descoberta de conhecimento. Posteriormente, Petrovic e Qu ([31]) enriqueceram o sistema com a utilização de busca tabu e *hill-climbing*.

Uma outra aplicação precursora de hiper-heurísticas a um problema de agendamento - desta vez, um caso real - pode ser encontrada em [22]. O problema consistia no transporte de galinhas vivas das fazendas às fábricas de processamento, de forma a atender aos pedidos de revendedores, que poderiam ser alterados todos os dias. Mais especificamente, a tarefa era agendar, sempre que necessário, o trabalho dos veículos de transporte disponíveis. Algumas das muitas restrições envolvidas eram: as galinhas não podem ser entregues muito antes dos horários solicitados pelas fábricas; os veículos estão disponíveis em diferentes dias e horários e têm capacidades distintas; as fazendas não podem ser visitadas em qualquer ordem.

A solução desenvolvida por Hart et al. para o problema acima emprega dois algoritmos genéticos, sendo o primeiro deles uma hiper-heurística em que cada cromossomo é composto por uma permutação dos pedidos e duas seqüências de decisões. A primeira

seqüência determina a divisão dos pedidos em tarefas de transporte, enquanto a segunda aloca veículos para cada uma dessas tarefas. O papel do segundo algoritmo genético é, partindo das associações criadas no passo anterior, estipular os horários de chegada de cada veículo às fábricas. Em uma terceira e última fase, o algoritmo construía uma agenda para os horários de partida dos veículos. Os resultados obtidos por esta estratégia foram satisfatórios, mas as mesmas ressalvas feitas à proposta de Fang et al. (1994) se aplicam: o algoritmo é altamente específico e o número de genes dos cromossomos cresce linearmente com o tamanho da instância do problema.

Em 1999, Ross et al. ([35]) desenvolveram uma hiper-heurística baseada em um algoritmo genético cujo tamanho de cromossomo já não era suscetível ao tamanho da instância. O problema tratado era a construção da grade horária de provas de uma universidade, levando em conta diversas restrições. Os exames podem ocorrer em alguns intervalos de tempo fixos e existem salas de vários tamanhos. Dois exames não podem ocorrer ao mesmo tempo se algum estudante realizará ambos. Dois exames podem ocorrer na mesma sala, ao mesmo tempo, caso a sala tenha capacidade suficiente. Alguns exames têm suas próprias restrições de horários. É desejável que um aluno nunca realize dois exames consecutivos e que os exames de maior porte sejam realizados antes.

O algoritmo desenvolvido em [35] para operar diretamente sobre o problema possui duas fases. A primeira delas repete, enquanto uma certa condição X não for satisfeita, a aplicação de uma heurística H_1 para selecionar um evento e de uma heurística H_2 para determinar um horário para a realização do evento selecionado. A segunda fase assemelha-se à anterior, exceto pela condição de término - obviamente - e pela substituição das heurísticas H_1 e H_2 por, respectivamente, heurísticas H_3 e H_4. A hiper-heurística aparece como um algoritmo genético que seleciona as quatro heurísticas acima, além da condição de término X. As restrições leves - *soft constraints* ou restrições desejáveis - são tratadas pelas heurísticas de baixo nível. A aptidão de cada cromossomo é medida através da construção direta da grade horária que ele representa. Apesar desta última característica, os resultados obtidos por Ross et al. foram satisfatórios.

3.3 Uma arquitetura robusta

Os métodos hiper-heurísticos acima têm em comum o fato de que precisariam de

alterações em todos os níveis para serem adaptados a novos problemas. Esses algoritmos, portanto, vão de encontro ao ideal de hiper-heurísticas como métodos bem definidos e robustos que, na aplicação a diferentes problemas, exigem apenas substituições no nível mais baixo, ou seja, na operação heurística direta sobre o problema específico. Uma condição necessária para que esse ideal seja atingido é, portanto, que seja definida pelo menos uma arquitetura na qual a construção de métodos hiper-heurísticos possa estar baseada. De fato, Ross et al. sintetizaram, em [34], uma arquitetura que houvera sido empregada em alguns desenvolvimentos. Apresenta-la-emos a seguir.

A hiper-heurística deve operar no nível mais alto, sem necessariamente conhecer o problema cuja resolução é o objetivo final. Este nível deve ser parametrizado com um conjunto de heurísticas que operam diretamente sobre o problema, além de uma função capaz de avaliar as soluções propostas por cada uma dessas heurísticas. A operação interna desses parâmetros também não é conhecida pela hiper-heurística. Desta maneira, o tratamento a um novo problema pode ser realizado apenas com a substituição dos parâmetros.

Naturalmente, deve haver uma interface bem definida entre a hiper-heurística e as heurísticas de baixo nível. Caso contrário, a hiper-heurística teria de conhecer a interface oferecida por cada heurística, o que potencialmente exigiria alterações freqüentes no nível mais alto. Além disso, uma interface padronizada torna mais simples a tarefa de se transferirem informações - independentes do domínio do problema - entre os níveis. Por exemplo, a hiper-heurística tem a possibilidade, ao chamar uma heurística, de fazer com que esta conheça o tempo disponível para a sua execução. Com isso, cada heurística pode executar até que o tempo tenha terminado e reportar os resultados obtidos, facilitando uma eventual análise comparativa a ser realizada pela hiper-heurística na etapa corrente.

Uma hiper-heurística deve armazenar diversas informações independentes do problema que permitam, em cada etapa, decidir qual heurística será utilizada. Algumas informações úteis podem ser: o tempo de CPU tomado na última chamada a cada heurística, a mudança no custo da solução após a última chamada de cada heurística, o tempo decorrido desde que cada heurística foi executada pela última vez, o ganho médio de qualidade obtido quando determinado par de heurísticas é utilizado em seqüência etc. Com informações desse tipo, a hiper-heurística pode adotar, em seu processo de decisão,

critérios como eficiência e eficácia - possivelmente, levando em conta pares ou trios de heurísticas -, tempo de inatividade etc.

A construção de uma hiper-heurística deve ser um processo bastante cuidadoso, sob pena de estar demasiado suscetível às falhas das heurísticas de baixo nível. Como ilustração mais clara desse processo de desenvolvimento, apresentamos, a seguir, alguns exemplos concretos.

3.4 Exemplo 1: escalonamento de professores

Em 2002, Cowling et al. ([13]) desenvolveram uma hiper-heurística para o tratamento de um problema de escalonamento de professores. Neste problema, são dados:

- 25 docentes.
- 10 centros de treinamento - que chamaremos apenas de localidades -, cada um com um número limitado de salas.
- Uma matriz que, a cada docente *d* e localidade *l*, associa um valor de *penalidade*, que indica quão indesejável é que o docente *d* precise ministrar um curso em *l*.
- 60 vagas para a grade horária.
- Um conjunto de cursos. A cada curso *c* estão associados: um conjunto de docentes competentes para ministrá-lo; um intervalo dentro do qual deve estar contido o início da execução de *c*; um conjunto de localidades dentre as quais deve ser selecionado o local de aplicação de *c*. Além disso, cada curso tem uma duração, que pode variar entre 1 e 5 vagas da grade horária, e um valor que indica a sua *prioridade* - quanto maior este valor, maior é a importância de se ministrar o curso.

Sejam: *W* a soma das prioridades dos cursos realizados; *D* a soma das penalidades aplicadas. O objetivo é maximizar o valor de *W* - *D*. Fora as restrições diretamente implicadas pelos dados acima, têm-se:

- Um docente não pode ministrar dois cursos simultaneamente.
- Cada docente pode trabalhar em no máximo 60% dos horários disponíveis.

Inicialmente, para fins de comparação, foram implementadas duas meta-heurísticas. A primeira delas é um algoritmo genético em que cada cromossomo possui 25 genes,

associados aos professores. Cada gene contém detalhes sobre as atividades realizadas pelo respectivo professor. O operador de cruzamento seleciona aleatoriamente dois genes e permuta os conjuntos de atividades entre os dois respectivos professores - naturalmente, é necessário, após essa operação, remover os conflitos potencialmente criados. O operador de mutação seleciona um gene aleatoriamente e acrescenta uma atividade à lista do respectivo docente. A população inicial é obtida a partir dos agendamentos criados por um algoritmo de *hill-climbing*. Os valores mais adequados para a taxa de cruzamento (0,6), a taxa de mutação (0,03), o tamanho da população (30) e o número de gerações (100) foram selecionados empiricamente. Na passagem entre duas gerações, os 10 melhores indivíduos são sempre mantidos (elitismo).

A segunda meta-heurística implementada por Cowling et al., bem como a hiper-heurística, utiliza-se de um conjunto de heurísticas simples, que podem ser divididas em três classes: heurísticas de *adição*, heurísticas de *intercâmbio* e heurísticas de *adição-remoção*. Existem cinco heurísticas de adição:

- *Adiciona-primeiro*: seleciona o curso não agendado de maior prioridade e encontra o primeiro par professor-localidade que pode ser associado ao curso.
- *Adiciona-aleatório*: semelhante à heurística acima, mas considera os professores e localidades em ordem aleatória.
- *Adiciona-melhor*: semelhante às heurísticas acima, mas seleciona o par professor-localidade que resulte na menor penalidade.
- *Adiciona-primeiro-aumento*: seleciona aleatoriamente um curso não agendado e encontra o primeiro par professor-localidade que resulte em um aumento na função objetivo.
- *Adiciona-maior-aumento*: semelhante à heurística acima, mas seleciona o par professor-localidade que resulte no maior aumento possível da função objetivo.

As heurísticas de intercâmbio são quatro:

- *Permuta-primeiro* e *permuta-aleatório*: análogas a *adiciona-primeiro* e *adiciona-aleatório*, mas, na presença de conflitos que possam tornar a solução inviável, verificam permutações (possivelmente todas elas) entre cursos, descobrindo se alguma delas resolve os conflitos.

- *Permuta-primeiro-aumento* e *permuta-maior-aumento*: análogas a *adiciona-primeiro-aumento* e *adiciona-maior-aumento*, com resolução de conflitos semelhante à das heurísticas acima.

As heurísticas de adição-remoção consideram os cursos não agendados em ordem decrescente de prioridade. Se o curso escolhido para adição causar conflitos com outros cursos, mas, mesmo com a remoção dos cursos conflitantes, causar um aumento na função objetivo, realizam-se as remoções necessárias e a adição do curso escolhido. Há três heurísticas desse tipo:

- *Adiciona-remove-primeiro*: seleciona, para o curso escolhido, o primeiro par professor-localidade que possa ser associado a ele.
- *Adiciona-remove-aleatório*: semelhante à heurística acima, mas considera os professores e localidades em ordem aleatória.
- *Adiciona-remove-melhor*: semelhante às heurísticas acima, mas seleciona o par professor-localidade que resulte em menor penalidade.

A segunda meta-heurística implementada é um algoritmo memético, que estende o algoritmo genético descrito acima. A busca por entornos é aplicada em cada geração, após as operações de cruzamento e mutação. Para cada cromossomo, seleciona-se aleatoriamente uma heurística de adição e se utiliza-a no agendamento de um curso. Se não for possível adicionar o curso selecionado, seleciona-se aleatoriamente uma das heurísticas de intercâmbio para aplicação. Se ainda não for possível adicionar o curso, aplica-se uma heurística de adição-remoção aleatoriamente escolhida. Este processo é repetido até que todos os cursos não agendados tenham sido considerados para inserção.

A hiper-heurística também é um algoritmo genético, mas cada cromossomo é uma seqüência de números inteiros no intervalo [0, 11]. Os cromossomos possuem tamanho fixo e cada gene representa uma das 12 heurísticas simples descritas acima. Cada cromossomo representa a aplicação das heurísticas simples ao problema, na ordem dada. Resultados empíricos sugerem a utilização de uma taxa de cruzamento de 0,6, uma taxa de mutação de 0,1, populações de tamanho 30 e 100 gerações. Porém, os autores utilizaram 200 gerações, com o intuito de observar as mudanças subseqüentes na distribuição das heurísticas. Para a construção da população inicial, foi adotado um processo simples que gera aleatoriamente números inteiros no intervalo [0, 11]. É

interessante notar que a hiper-heurística se mostrou robusta em relação às alterações de parâmetros realizadas nos testes.

Quatro versões da hiper-heurística foram adotadas na comparação com outros métodos. As diferenças entre essas versões residem na adaptabilidade e na função de aptidão. As duas versões adaptativas elevam o valor da taxa de mutação e diminuem o valor da taxa de cruzamento sempre que não há aumento na aptidão média da população em três gerações seguidas. Quando, por outro lado, há aumento de qualidade em três gerações seguidas, a taxa de mutação diminui e a taxa de cruzamento se eleva. O aumento no valor de um parâmetro A é realizado da seguinte maneira: $A_{novo} = (A_{antigo} + 1)/2$. A diminuição consiste na divisão do valor de um parâmetro por 2. Duas versões adotam como função de aptidão o valor objetivo da solução encontrada pela aplicação da seqüência de heurísticas sugerida pelo cromossomo à melhor solução já encontrada em todo o processo. As outras duas versões levam a eficiência em consideração, dividindo o valor objetivo (calculado como descrito acima) pelo tempo de CPU consumido pela aplicação das heurísticas na seqüência sugerida pelo cromossomo.

Concluindo, os testes realizados por Cowling et al. comparam os resultados de diversos métodos:

- As 12 heurísticas de baixo nível descritas acima. O critério de parada para cada uma delas é a verificação de que todos cursos não agendados já foram levados em consideração.
- Cada uma das quatro hiper-heurísticas.
- O algoritmo genético que opera diretamente sobre o problema.
- O algoritmo memético.
- Cinco heurísticas (H_1 a H_5) que foram os respectivos resultados de cinco execuções da hiper-heurística mais simples - a versão não adaptativa cuja função de aptidão não leva em conta o tempo de CPU - em uma instância razoavelmente complexa do problema. Cada uma dessas cinco heurísticas corresponde, portanto, a um cromossomo do tipo utilizado pelas hiper-heurísticas.

Foram utilizadas cinco instâncias do problema, que diferem no grau de complexidade, associado principalmente ao número médio de professores que podem ministrar cada

curso e ao número médio de localidades em que cada curso pode ser ministrado. Na instância mais complexa, cada curso pode ser ministrado em apenas uma localidade, por um dentre, no máximo, cinco docentes. Na instância mais simples, cada curso pode ser ministrado por qualquer professor, em qualquer localidade. As instâncias são baseadas em um caso real ocorrido em uma instituição financeira de grande porte.

Os resultados mais importantes observados foram:

- As heurísticas H_1 a H_5 se mostraram superiores às 12 heurísticas mais simples, em todas as instâncias do problema. Este resultado indica que a hiper-heurística explorou as qualidades das heurísticas simples de maneira satisfatória.
- Naturalmente, as soluções encontradas pelas 12 heurísticas simples são muito inferiores aos resultados das quatro versões da hiper-heurística - que, no entanto, demandam um tempo de CPU várias vezes maior.
- Ambas as meta-heurísticas que operam diretamente sobre o problema forneceram, em maior tempo de CPU, soluções consideravelmente inferiores às das hiper-heurísticas.
- A freqüência de utilização de cada heurística simples varia bastante entre as hiper-heurísticas.
- A hiper-heurística mais simples obteve os melhores resultados em todas as instâncias, excetuando-se a menos complexa - esta foi tratada com maior desenvoltura pela hiper-heurística que utiliza parâmetros adaptativos e adota a própria função objetivo como função de aptidão.

Os mesmos autores desenvolveram ([12]), ainda em 2002, versões das hiper-heurísticas acima cujos cromossomos possuem tamanhos variáveis. O objetivo desta nova estratégia é óbvio: aumentar a flexibilidade e a autonomia das hiper-heurísticas, permitindo-lhes construir seqüências de heurísticas de baixo nível utilizando métodos que não estejam atrelados à necessidade de respeitar o tamanho corrente dos cromossomos. Idealmente, o tamanho de cromossomo mais adequado será determinado sem a necessidade de se realizarem experimentos manualmente. A principal hipótese adotada pelas hiper-heurísticas é a de que uma seqüência de genes que, em determinado cromossomo, interfere positivamente no resultado oferecido pelo cromossomo tem grandes chances de causar o mesmo impacto ao ser inserida em algum trecho de outro cromossomo.

Os mesmos operadores de cruzamento e mutação utilizados em [13] são também adotados por Cowling et al. nas novas estratégias. Porém, três outros operadores foram introduzidos. Um novo operador de cruzamento seleciona a melhor seqüência de genes - a que tem melhor impacto na função objetivo - de cada cromossomo e permuta essas seqüências entre os cromossomos. Uma nova mutação seleciona a pior seqüência de genes - a que tem o pior impacto na função objetivo ou a mais longa dentre as que não promovem aumento de qualidade - e a remove do cromossomo. Outra mutação seleciona a melhor seqüência de genes de um cromossomo - aleatoriamente escolhido - e a insere, em uma posição aleatória, no cromossomo selecionado para sofrer mutação.

Devido à presença dos novos operadores, as hiper-heurísticas utilizam um valor de penalidade para evitar a presença de cromossomos demasiado grandes. Cromossomos com maior valor de penalidade terão menor probabilidade de serem selecionados pelos operadores. Este valor é calculado pela expressão $T \cdot C/M$, onde T é o tamanho do cromossomo, C é o tempo de CPU necessário para a avaliação do cromossomo e M é o aumento de qualidade resultante da aplicação da seqüência de heurísticas sugerida pelo cromossomo.

Os resultados exibidos por Cowling et al. ([12]) referem-se a quatro implementações de hiper-heurísticas com tamanho adaptativo de cromossomo, que são comparadas às quatro respectivas hiper-heurísticas que lhes deram origem, às heurísticas H1 a H5 e às meta-heurísticas que operam diretamente sobre o problema. Além disso, foi implementada e avaliada uma versão que, dos novos operadores, utiliza apenas a mutação - desprezando, portanto, o novo operador de cruzamento. Utilizaram-se, nos experimentos, as mesmas instâncias outrora adotadas. As principais observações dos autores estão listadas abaixo:

- Cada versão da hiper-heurística com tamanho adaptativo de cromossomo superou, na maioria dos casos, a respectiva versão com cromossomos de tamanho fixo.
- A hiper-heurística que forneceu as melhores soluções é justamente a nova versão da hiper-heurística que fornecera as melhores soluções em [13].
- A versão que não utiliza o novo operador de cruzamento obteve resultados comparáveis (embora inferiores) aos das demais hiper-heurísticas, com menor consumo de tempo.

3.5 Exemplo 2: feira comercial

Em 2001, Cowling, Kendall e Soubeiga ([15]) desenvolveram uma hiper-heurística cujo funcionamento está mais próximo da arquitetura comentada acima (Ross et al. - [34]). Desta vez, o problema era a definição de uma agenda para um evento comercial, promovido por uma companhia, envolvendo *fornecedores* - representantes de empresas interessadas em vender produtos ou serviços - e *delegados* - representantes de empresas potencialmente interessadas nos produtos e serviços dos fornecedores. Cada fornecedor paga uma taxa de participação e indica uma lista de delegados que gostaria de encontrar, classificando cada possível encontro como *prioritário* ou *não-prioritário*. Além dos encontros, são organizados *seminários*, que promovem a interação entre delegados. Cada delegado indica uma lista com os seminários dos quais gostaria de participar. A participação dos delegados em todos os seminários desejados deve ser garantida, a não ser que o delegado não seja incluído no evento. Existem 43 fornecedores, 99 potenciais delegados, 12 seminários e 24 horários disponíveis. Cada encontro ocupa um horário, enquanto os seminários ocupam três horários cada. O objetivo é agendar os encontros, ou seja, criar tuplas (delegado, fornecedor, horário) satisfazendo as seguintes restrições (as duas últimas são apenas desejáveis):

- Cada delegado deve participar de no máximo 12 encontros.
- Um delegado não pode participar de duas atividades (seminário ou encontro) ao mesmo tempo.
- Um fornecedor não pode participar de dois encontros ao mesmo tempo.
- Cada fornecedor deve ter ao menos 17 encontros prioritários.
- Cada fornecedor deve ter ao menos 20 encontros.

A função objetivo a ser minimizada, no problema abordado em [15], pode ser definida com a fórmula $E = B + 0{,}05 \cdot C + 8 \cdot H$, onde B é a soma das penalidades associadas, para cada fornecedor, ao não cumprimento da penúltima restrição acima, C é a soma das penalidades relacionadas à última restrição e $H = d - 72$, sendo d o número de delegados a serem convidados para o evento. O termo H é importante devido ao fato de que a presença dos delegados implica gastos, para a companhia organizadora do evento, com hospedagem e transporte. O valor 72 é um limite inferior para o número de delegados

quando cada fornecedor participa de 20 encontros. O algoritmo guloso inicialmente desenvolvido pela companhia organizadora do evento descrito acima fornece uma solução de custo 444,43, enquanto o algoritmo guloso desenvolvido por Cowling et al. resulta em custo 225,55. Esta última solução é utilizada como ponto de partida para a hiper-heurística descrita a seguir.

Nesta implementação, a hiper-heurística é parametrizada com o conjunto de *critérios* - e respectivos pesos - que compõem a função objetivo. Para o problema em mãos, esses critérios são as funções *B*, *C* e *H*, com pesos, respectivamente, 1, 0,05 e 8, dados pela fórmula da função objetivo. A hiper-heurística, então, executa um processo iterativo que, em cada passo, seleciona um critério *c* e decide qual heurística de baixo nível deve ser aplicada para aumentar a qualidade da solução em relação a *c*. Seja H_k a heurística que foi executada antes do passo atual. A seleção da próxima heurística basear-se-á em três fatores históricos: o desempenho (f_{1c}) de cada heurística no que diz respeito às mudanças de avaliação da solução em relação a *c*; o desempenho (f_{2c}) de cada heurística (em relação a *c*) quando executada logo após H_k; o tempo (f_3) decorrido desde que cada heurística foi executada pela última vez. Claramente, os dois primeiros fatores visam à intensificação da busca, enquanto o último focaliza a diversificação. Cada um desses fatores tem um grau de influência distinto no processo de decisão, o que é representado através da atribuição de um peso dinâmico a cada um deles. Mais especificamente, a função de decisão, para cada critério *c*, pode ser escrita como: $f_c(H_i) = \alpha \cdot f_{1c}(H_i) + \beta \cdot f_{2c}(H_i, H_k) + \delta \cdot f_3$, sendo H_i uma heurística candidata.

Como observado acima, o peso de cada fator histórico no processo de decisão da hiper-heurística pode ser alterado durante a execução. Quando a heurística selecionada aumenta a qualidade da solução, o peso de um fator de intensificação sofre elevação proporcional ao ganho de qualidade. De forma análoga, esse peso é reduzido na situação oposta. O fator cujo peso será alterado é, em qualquer dos casos acima, o que tem maior influência no processo de decisão, ou seja, o fator de maior peso corrente. O fator de diversificação tem seu peso elevado quando, durante um certo número de iterações (que corresponde ao número de heurísticas de baixo nível existentes), não há progressos em relação ao critério considerado. De forma análoga, o peso deste fator é decrementado quando há seguido aumento de qualidade.

A hiper-heurística se utiliza de 10 heurísticas de baixo nível, que se assemelham fortemente aos operadores de construção de vizinhança utilizados pelas meta-heurísticas de busca por entornos:

- Remover um delegado.
- Duas diferentes heurísticas para: selecionar um delegado, remover um de seus encontros e adicionar outro.
- Adicionar um delegado.
- Adicionar um encontro para um fornecedor insatisfeito.
- Adicionar um encontro para um fornecedor que esteja insatisfeito em relação aos seus encontros prioritários.
- Remover encontros excedentes de um fornecedor.
- Remover um encontro e adicionar outro para um fornecedor.
- Duas diferentes heurísticas para: selecionar um fornecedor *f* que esteja insatisfeito em relação a seus encontros prioritários, remover um encontro de *f* e adicionar outro.

Versões iniciais da hiper-heurística consideravam os critérios componentes da função objetivo de forma cíclica, procurando aumentar a qualidade em relação a um deles a cada ponto de decisão. Como há conflitos entre os critérios, o resultado ficou aquém do esperado (soluções de custo entre 62,15 e 89,5), uma vez que a hiper-heurística apresentava um comportamento cíclico em relação às alterações realizadas na solução. A estratégia efetivamente adotada, portanto, foi selecionar aleatoriamente, em cada ponto de decisão, o critério a ser considerado no próximo passo. Para diminuir ainda mais a probabilidade de ocorrerem ciclos, a probabilidade de que cada critério seja selecionado é proporcional ao peso do critério na função objetivo.

Para fins de comparação, desenvolveram-se 4 hiper-heurísticas que operam de forma semelhante às meta-heurísticas de busca por entornos. A seguir, descrevemos o funcionamento de uma iteração de cada uma dessas hiper-heurísticas:

- *SimpleRandom*: seleciona uma heurística aleatoriamente e a aplica à solução corrente.
- *RandomDescent*: seleciona uma heurística aleatoriamente e a aplica repetidamente até que não haja ganho de qualidade.

- *RandomPerm*: cria aleatoriamente uma permutação das heurísticas e a aplica.
- *RandomPermDescent*: semelhante ao método acima, mas cada heurística é aplicada, repetidamente, até que não haja ganho de qualidade.

Para a implementação da hiper-heurística definitiva, os autores basearam-se no desempenho das hiper-heurísticas simples. Mais especificamente, os passos que objetivam a intensificação aplicam a heurística selecionada repetidamente, até que não haja diminuição de custo possível, enquanto os passos de diversificação realizam chamadas únicas. Para a execução dos experimentos, todas as hiper-heurísticas foram implementadas em duas variantes. Na variante *OI*, apenas os passos que promovem aumento de qualidade são aceitos, enquanto, na variante *AM*, todos os passos selecionados pelo critério de decisão são aceitos. A mesma instância do problema foi utilizada em todos os experimentos e cada algoritmo - incluindo as estratégias gulosas da companhia e dos autores - foi executado 10 vezes, com 30 minutos para cada execução. Para cada algoritmo, o valor considerado nas comparações foi o custo médio das soluções encontradas em cada uma de suas execuções. Dentre as observações dos autores, destacam-se:

- Os melhores resultados, dentre as hiper-heurísticas mais simples, foram obtidos pelas variantes *AM* dos métodos que realizam otimização local em cada iteração.
- A hiper-heurística principal, em sua variante *AM*, obteve resultados significativamente superiores aos dos demais métodos.
- Os resultados obtidos pela variante *OI* da hiper-heurística principal, apesar de inferiores aos da variante *AM* e não muito distantes do melhor resultado obtido pelos demais métodos, superaram todas as hiper-heurísticas simples, bem como os algoritmos gulosos.
- Todas as hiper-heurísticas simples superaram os algoritmos gulosos, embora os métodos *SimpleRandom-AM* e *RandomPerm-AM* não tenham oferecido ganhos significativos.

3.6 Exemplo 3: agendamento de apresentações

Os mesmos autores do exemplo acima, com o objetivo de demonstrar a robustez do

método, aplicaram ([14]) a mesma hiper-heurística a um problema de agendamento de apresentações. Com menos de três semanas de desenvolvimento, foram obtidas soluções de qualidade também para este novo problema, que consiste no agendamento de apresentações de projetos em uma universidade britânica. Cada aluno no último ano da graduação desenvolve um projeto, supervisionado por um docente. Tal projeto deve, então, ser apresentado perante uma banca composta por três docentes (incluindo, possivelmente, o próprio supervisor): *primeiro avaliador*, *segundo avaliador* e *observador*. As apresentações são organizadas em sessões e cada par *(sessão, sala)* deve conter no máximo 6 apresentações. O objetivo, portanto, é construir tuplas na forma *(aluno, 1º avaliador, 2º avaliador, observador, sessão, sala)*, sob a restrição adicional de que nenhum docente deve, em uma mesma sessão, compor banca em salas distintas. O valor a ser minimizado pode ser expresso como $E = 0{,}5 \cdot A + B + 0{,}3 \cdot C - D$, onde:

- A é a variância do número de apresentações por docente.
- B é a variância do número de sessões por docente.
- C é a variância do número de sessões "ruins" por docente. Uma sessão é "ruim" se ocorre antes das 10h ou após as 16h.
- D é um número que cresce com o número de apresentações em que o supervisor faz parte da banca e com o grau de interesse de cada docente nas áreas dos projetos em que compõe banca.

Na instância do problema acima visada por Cowling et al., existem 26 docentes e apenas duas salas disponíveis para 80 sessões, que devem comportar 151 apresentações. Inicialmente, foi desenvolvida uma heurística construtiva simples que, em cada iteração, seleciona um trio de docentes (os docentes são considerados ciclicamente, de forma a satisfazer os objetivos A e B), destinando-o a 6 apresentações (se possível) no primeiro par *(sessão, sala)* disponível, considerando-se primeiramente as sessões que não sejam "ruins" (objetivo C) e as apresentações cujos supervisores pertençam à banca e cujas áreas sejam mais interessantes para os membros da banca. A solução fornecida por esta heurística foi o ponto de partida de todas as hiper-heurísticas desenvolvidas em [14].

As primeiras hiper-heurísticas utilizadas por Cowling et al. para o problema das apresentações foram exatamente as 4 hiper-heurísticas simples (em suas versões *AM* e

OI) utilizadas no problema da feira comercial. Naturalmente, foi necessário desenvolver heurísticas de baixo nível que pudessem lidar com o domínio do novo problema. De fato, as 8 seguintes heurísticas desse tipo foram implementadas:

- N_1: seleciona um docente d_1 e uma sessão s (que contenha um subconjunto não vazio A_1 das apresentações de que d_1 participa) e substitui, em todas as apresentações de A_1, o docente d_1 pelo docente d_2, sendo d_2 um docente qualquer que não esteja escalado para apresentações na sessão s.
- N_2: semelhante a N_1, mas d_1 será o docente que está escalado para o maior número de sessões.
- N_3: semelhante a N_2, mas s será a sessão de que d_1 compõe banca no menor número de apresentações. Além disso, d_2 pode estar escalado para apresentações de s.
- N_4: seleciona aleatoriamente uma apresentação a e uma sessão $s_2 \neq s_1$ (onde s_1 é a sessão para a qual a estava agendada), agendando a para a sessão s_2, em uma sala também escolhida aleatoriamente. Naturalmente, a será removida de s_1.
- N_5: seja a_{ijks} o número de apresentações associadas à banca *(i, j, k)* na sessão s. Esta heurística funciona de maneira semelhante a N_4, mas a será uma apresentação cujo par *(sessão, banca)* minimize a_{ijks}.
- N_6: semelhante a N_5, mas s_2 será uma sessão na qual pelo menos um dos membros da banca de a já esteja presente.
- N_7: seleciona aleatoriamente duas apresentações a_1 e a_2 e permuta o 2º avaliador de a1 com o observador de a2. Essa troca só ocorre se não envolver a remoção de um supervisor.
- N_8: semelhante a N_7, mas permuta o 1º avaliador de a_1 com o 2º avaliador de a_2.

Três hiper-heurísticas principais foram aplicadas ao problema acima. A primeira delas - HH_1 - é exatamente a estratégia adotada em [15] (parametrizada com os novos componentes da função objetivo e as novas heurísticas simples). Uma variação - HH_2 -, também implementada, dessa hiper-heurística estipula valores fixos para os pesos dos fatores históricos no processo de decisão, tanto na versão *AM* ($\alpha = \beta = 0,1$ e $\delta = 2,5$), quanto na versão *OI* ($\alpha = \beta = 0,1$ e $\delta = 1,5$). A última hiper-heurística desenvolvida -

HH_3 - assemelha-se às duas anteriores, adotando, da primeira, os pesos adaptativos para os fatores históricos. Esta terceira hiper-heurística difere das anteriores por receber um único critério (E) em relação ao qual a otimização deve ser realizada (ou seja: não é necessário parametrizar a hiper-heurística com as funções A, B, C, D e seus respectivos pesos).

Os experimentos realizados em [14] visam à comparação entre as 3 hiper-heurísticas principais, as 4 hiper-heurísticas simples, uma solução manual e a heurística construtiva simples elaborada pelos autores. Todas as hiper-heurísticas foram executadas em suas versões *AM* e *OI* e o critério de parada para todos os algoritmos - cada um executado 10 vezes, para observação da qualidade média de cada método - foi o consumo de 10 minutos de tempo de CPU. Dentre os resultados observados, destacam-se:

- Como esperado, todas as hiper-heurísticas obtiveram resultados muito superiores à solução manual e à solução fornecida pela heurística construtiva simples.
- Dentre as hiper-heurísticas simples, *RandomDescent* e *RandomPermDescent* (tanto nas versões *AM* quanto nas versões *OI*) foram os destaques.
- A solução de maior qualidade foi encontrada pela hiper-heurística HH_3, em sua versão *AM*. Além disso, HH_3 foi a único método a se destacar consideravelmente em relação às hiper-heurísticas simples.
- Contrariando as expectativas, a hiper-heurística HH_2 mostrou-se superior a HH_1 na instância considerada do problema.
- Embora N_3 seja a heurística mais utilizada tanto por HH_1 quanto por HH_3, houve diferenças consideráveis entre a freqüência de utilização das heurísticas simples por essas duas hiper-heurísticas. Enquanto N_3, N_2 e N_6 são as mais utilizadas por HH_3, N_3, N_1 e N_8 são as melhores para HH_1. Além disso, a importância de N_4, N_5 e N_7 é aparentemente maior na hiper-heurística HH_1.
- De forma geral, as hiper-heurísticas principais foram capazes de detectar a maior qualidade da heurística N_3 em relação a N_1 e N_2 e da heurística N_6 em relação a N_4 e N_5.
- As hiper-heurísticas também foram executadas partindo da solução construída manualmente. Neste caso, a performance relativa entre as hiper-heurísticas não foi

alterada significativamente. Porém, de forma geral, a qualidade das soluções encontradas ficou muito abaixo da média anterior, o que demonstra uma fragilidade dos métodos implementados quando, assistidos pelo conjunto dado de heurísticas simples, partem de uma região desfavorável do espaço de busca.

- Os autores compararam também o desempenho de *HH1* e *HH3* para o problema anterior (feira comercial) e, neste caso, obtiveram melhores resultados com HH_1, o que sugere um impacto considerável, na velocidade de evolução do algoritmo HH_1, do número de critérios componentes da função de avaliação.

4 Conclusão

As meta-heurísticas são técnicas de qualidade para a resolução de problemas complexos, mas implementações realmente eficazes e eficientes desses métodos normalmente requerem fortes conhecimentos específicos sobre o problema que se pretende tratar, além de dificilmente serem robustas em relação a alterações de domínio.

As hiper-heurísticas foram propostas com o objetivo de viabilizarem a construção de algoritmos robustos, possibilitando a utilização (com adaptações simples) de métodos de qualidade além do meio acadêmico e das empresas providas com altos recursos intelectuais. As pesquisas realizadas até aqui indicam fortes possibilidades de sucesso dos métodos hiper-heurísticos, embora a popularização e larga experimentação dessa classe de heurísticas ainda seja esperada para os próximos anos.

Referências

[1]: GLOVER, F., LAGUNA, M. *Tabu Search*, Kluwer, 1997.

[2]: MINTON, S. *Learning Search Control Knowledge:* An Explanation-based Approach. Kluwer, 1988.

[3]: RUSSELL, S., NORVIG, P. *Artificial Intelligence*: A Modern Approach. Prentice Hall, 1995.

[4]: PAPADIMITRIOU, C. H., STEIGLITZ, K. *Combinatorial Optimization*: Algorithms and Complexity. Minneola: Dover Publications, 1998.

[5]: ZIMMERMANN, H. J. *Fuzzy Sets Theory and Its Applications*. Kluwer Academic, 1996.

[6]: BATTITI, R. Reactive search: towards self-tuning heuristics. In RAYWARD-SMITH, V. J. et al. *Modern heuristic search methods*. Wiley, 1996. p. 61-83.

[7]: BATTITI, R., TECCHIOLLI, G. The reactive tabu search. In *ORSA Journal on Computing*, 1994. v.6, p. 126-140.

[8]: BLANCO, A., PELTA, D., VERDEGAY, J. L. FANS: una heurística basada en conjuntos difusos para problemas de optimización. In *Revista Iberoamericana de Inteligencia Artificial*. Asociación Española de Inteligencia Artificial, 2003. v. 2, n. 19.

[9]: BOETTCHER, S., PERCUS, A. G. Nature's way of optimizing. In *Artificial Intelligence*. 2000. v. 119. p. 275-286.

[10]: BRÄYSY, O. et al. *A Threshold Accepting Metaheuristic for the Vehicle Routing Problem with Time Windows*. Internal Report STF42 A02014 SINTEF Applied Mathematics. Oslo: 2002. Department of Optimization.

[11]: BURKE, E. et al. Knowledge Discovery in a Hyperheuristic for Course Timetabling using Case Based Reasoning. In *Proceedings of the Fourth International Conference on the Practice and Theory of Automated Timetabling (PATAT'02)*. Ghent: ago. 2002.

[12]: COWLING, P., KENDALL, G. HAN, L. *An Adaptive Length Chromosome*

Hyperheuristic Genetic Algorithm for a Trainer Scheduling Problem. Technical Report NOTTCS-TR-2002-5 (submitted to SEAL 2002 Conference). School of Computer Science & IT, University of Nottingham, 2002.

[13]: COWLING, P., KENDALL, G. HAN, L. An Investigation of a Hyperheuristic Genetic Algorithm Applied to a Trainer Scheduling Problem. In *Proceedings of Congress on Evolutionary Computation (CEC2002)*. Honolulu: 12-17 mai. 2002. p. 1185-1190.

[14]: COWLING, P., KENDALL, G., SOUBEIGA, E. Hyperheuristics: A Tool for Rapid Prototyping in Scheduling and Optimization. In CAGONI, S. et al. *LNCS 2279, Applications of Evolutionary Computing: Proceedings of Evo Workshops 2002*. Kinsale: Springer-Verlag, 3-4 abr. 2002. p. 1-10.

[15]: COWLING, P., KENDALL, G., SOUBEIGA, E. A Parameter-Free Hyperheuristic for Scheduling a Sales Summit. In *Proceedings of 4th Metaheuristics International Conference (MIC 2001*). Porto: 16-20 jul. 2001. p. 127-131.

[16]: DORIGO, M., STÜTZLE, T. The Ant Colony Optimization Metaheuristic: Algorithms, Applications, and Advances. In GLOVER, F., KOCHENBERGER, G. *Handbook on Metaheuristics*. Kluwer, 2003. Cap. 9.

[17]: FANG, H., ROSS, P., CORNE, D. A Promising Hybrid GA/Heuristic Approach for Open-Shop Scheduling Problems. In COHN, A. *Proceedings of ECAI 94: 11th European Conference on Artificial Intelligence*. John Wiley and Sons, 1994. p. 590-594.

[18]: GLOVER, F., MELIÁN, B. Búsqueda tabú. In *Revista Iberoamericana de Inteligencia Artificial*. Asociación Española de Inteligencia Artificial, 2003. v. 2, n. 19.

[19]: GRATCH, J., CHEIN, S., JONG, G. Learning search control knowledge for deep space network scheduling. In *Proceedings of the Tenth International Conference on Machine Learning*. 1993. p. 135-142.

[20]: GUIGNARD, M. Lagrangian Relaxation. In PARDALOS, P. M., RESENDE, M. G. C. *Handbook of Applied Optimization.* Oxford University Press, 2002. p. 465-474.

[21]: HANSEN, P., MLADENOVIC, N., PÉREZ, J. A. M. Búsqueda de entorno variable. In *Revista Iberoamericana de Inteligencia Artificial*. Asociación Española de Inteligencia Artificial, 2003. v. 2, n. 19.

[22]: HART, E., ROSS, P. M., NELSON, J. Solving a real-world problem using an evolving heuristically driven schedule builder. *Evolutionary Computation*. v. 6, n. 1. 1998. p. 61-80.

[23]: KHURI, S., *An Introduction to Genetic Algorithms* (lecture for the PhD students). 13 dez. 2000. University of Malaga.

[24]: LAGUNA, M., GLOVER, F., MARTÍ, R. Fundamentals of scatter search and path relinking. In *Control and Cybernetics*. 2000. v. 39, p. 653-684.

[25]: LAGUNA, M., GLOVER, F., MARTÍ, R. Scatter Search and Path Relinking: Foundations and Advanced Designs. In ONWUBOLU, G. C., BABU, B. V. *New Optimization Techniques in Engineering*. Springer, 2004.

[26]: LARRAÑAGA, P., LOZANO, J. A., MÜHLENBEIN, H. Algoritmos de estimación de distribuciones en problemas de optimización combinatoria. In *Revista Iberoamericana de Inteligencia Artificial*. Asociación Española de Inteligencia Artificial, 2003. v. 2, n. 19.

[27]: LIM, A., LIN, J., XIAO, F. Particle Swarm Optimization and Hill Climbing to Solve the Bandwidth Minimization Problem. In *The Fifth Metaheuristics International Conference* (MIC2003). Kyoto: 2003.

[28]: LOURENÇO, H. R., MARTIN, O., STÜTZLE, T. Iterated local search. In GLOVER, F., KOCHENBERGER, G. *Handbook of Metaheuristics*. Kluwer, 2003. Cap. 11.

[29]: MELIÁN, B., PÉREZ, J. A. M., VEGA, J. M. M. Metaheuristics: A Global View. In *Revista Iberoamericana de Inteligencia Artificial*. Asociación Española de Inteligencia Artificial, 2003. v. 2, n. 19.

[30]: MOSCATO, P., COTTA-PORRAS, C. Una introduccion a los algoritmos memeticos. In *Revista Iberoamericana de Inteligencia Artificial*. Asociación Española de Inteligencia Artificial, 2003. v. 2, n. 19.

[31]: PETROVIC, S., QU, R. Case-Based Reasoning as a Heuristic Selector in a Hyper-Heuristic for Course Timetabling. In *Proceedings of the Sixth International Conference on Knowledge-Based Intelligent Information & Engineering Systems (KES'2002)*. Crema: set. 2002.

[32]: REEVES, C. R. Genetic Algorithms. In GLOVER, F., KOCHENBERGER, G. *Handbook of Metaheuristics*. Kluwer, 2003. Cap. 3.

[33]: RESENDE, M. G. C., RIBEIRO, C. C. Greedy randomized adaptive search procedures. In GLOVER, F., KOCHENBERGER, G. *Handbook of Metaheuristics*. Kluwer, 2003.

[34]: ROSS, P. et al. Hyper-heuristics: An Emerging Direction in Modern Search Technology. In GLOVER, F., KOCHENBERGER, G. *Handbook of Metaheuristics*. Kluwer, 2003. p. 457-474.

[35]: TERASHIMA-MARÍN, H., ROSS, P. M., VALENZUELA-RENDÓN, M. Evolution of constraint satisfaction strategies in examination timetabling. In BANZHAF, W. et al. *Proceedings of the GECCO-99 Genetic and Evolutionary Computation Conference*. Morgan Kaufmann, 1999. p. 635-642.

[36]: VOUDOURIS, C., TSANG, E. P. K. Guided local search. In GLOVER, F., KOCHENBERGER, G. *Handbook on Metaheuristics*. Kluwer, 2003. Cap. 7.

[37]: *The Harwell-Boeing Sparse Matrix Collection*. (http://math.nist.gov/MatrixMarket/data/Harwell-Boeing)

[38]: *Metaheuristics Network*. (http://www.metaheuristics.net)

[39]: HU, X. *Particle Swarm Optimization*. (http://www.swarmintelligence.org)

[40]: POMEROY, P. *An Introduction to Particle Swarm Optimization*. 2003. (http://www.adaptiveview.com/articles/ipsop1.html)