MESTRADO EM
MÉTODOS QUANTITATIVOS EM GESTÃO

UMA ABORDAGEM ORIENTADA POR OBJECTOS PARA META-HEURÍSTICAS MULTIOBJECTIVO

JOÃO ALBERTO VIEIRA DE CAMPOS PEREIRA CLARO

FEVEREIRO DE 2002

Tese submetida para satisfação parcial

dos requisitos do programa de mestrado

em

Métodos Quantitativos em Gestão

Tese realizada sob a supervisão do

Prof. Doutor Jorge Manuel Pinho de Sousa

Professor Associado da

Faculdade de Engenharia da Universidade do Porto

Resumo

Nesta dissertação convergem algumas linhas de investigação na área das meta-heurísticas que, recentemente, têm vindo a ser objecto de particular atenção: a *flexibilização*, ou seja, a introdução de mecanismos de modificação de componentes e estratégias elementares, o desenvolvimento de abordagens orientadas por objectos, e a adaptação a contextos multiobjectivo. Esta convergência justifica-se pelo facto de as abordagens orientadas por objectos promoverem naturalmente a flexibilização, e pela constatação da inexistência, até ao momento, de abordagens orientadas por objectos para a área das meta-heurísticas multiobjectivo.

Foi feita uma análise e sistematização do domínio e, em particular, das meta-heurísticas multiobjectivo, com ênfase na perspectiva da flexibilização. Esta sistematização fundamenta a proposta de um *template* para pesquisa local multiobjectivo, e de um conjunto de estratégias genéricas de flexibilização. A análise realizada constitui a base para o desenvolvimento de um *framework* orientado por objectos, estruturado a partir do *template* proposto, com sucessivas extensões, para inclusão de meta-heurísticas "básicas" (PSA e MOTS*), e de estratégias genéricas de flexibilização (listas de candidatos, vizinhanças variáveis e hibridização e paralelização de alto nível).

A descrição e validação do *framework* é completada com a sua aplicação ao problema de escalonamento multiobjectivo de tarefas numa máquina, minimizando o atraso máximo, o número de tarefas atrasadas e a soma ponderada dos atrasos, e com a realização de um conjunto de testes computacionais.

As experiências realizadas demonstram o potencial da abordagem proposta. Permitem também concluir do interesse da utilização de estratégias genéricas de flexibilização e da adequação da abordagem proposta para concepção e implementação de meta-heurísticas multiobjectivo.

Abstract

Several research directions in the field of meta-heuristics, that have recently been the focus of significant attention, converge in this dissertation: the *flexibilisation of meta-heuristics*, i.e. the introduction of mechanisms for the modification of basic components and strategies, the development of object oriented approaches, and the adaptation to multiobjective contexts. This convergence is justified by the fact that the object oriented paradigm seems to be particularly well-suited for the design and implementation of flexibilisation based approaches, and by the fact that no object oriented approach has yet been proposed for multiobjective meta-heuristics.

A systematic analysis of the domain has been performed, with particular emphasis on multiobjective meta-heuristics and on flexibilisation. This analysis sets the foundation for the proposal of a multiobjective local search template, and a set of generic flexibilisation strategies. It also supports the development of an object oriented framework, structured around the proposed template, with a sequence of extensions that allow the inclusion of "basic" meta-heuristics (PSA and MOTS*), and generic flexibilisation strategies (candidate lists, neighbourhood variation and high level, parallel, hybridisation).

The description and validation of the framework is completed by applying it to the multiobjective single machine scheduling problem of minimising the number of late jobs, the maximum tardiness and the weighted tardiness, and by performing a set of computational experiments with that problem.

These experiments clearly show the interest of using generic flexibilisation strategies. They also show the potential of the proposed object oriented approach for the design and implementation of multiobjective meta-heuristics.

Palavras chave

Meta-heurísticas multiobjectivo

Estratégias genéricas de flexibilização em meta-heurísticas

Frameworks orientados por objectos

Escalonamento multiobjectivo de tarefas numa máquina

Keywords

Multiobjective meta-heuristics

Generic flexibilisation strategies in meta-heuristics

Object oriented frameworks

Multiobjective single machine scheduling

Agradecimentos

Ao Prof. Jorge Pinho de Sousa, por ser mais do que um professor e um orientador; por ser um mestre.

Ao Prof. Rui Guimarães, ao Prof. Jorge Pinho de Sousa, novamente, e ao Prof. José Fernando Oliveira, pelas oportunidades de docência e investigação que me proporcionaram, ao longo da realização deste trabalho.

À Ana Viana, pela disponibilidade constante para discutir questões que o trabalho foi levantando.

Ao Jorge e ao José António, pela amizade e pelo excepcional trabalho em equipa, na parte escolar do mestrado.

Aos meus pais e às minhas irmãs, aí onde estão as minhas raízes.

À Teresa, ao Daniel e ao Pedro, os meus mais-que-tudo.

A um Amor Supremo.

Em memória de minha mãe

Para o meu pai

Para a Teresa, o Daniel e o Pedro

« Not "Revelation" - 'tis - that waits,

But our unfurnished eyes - »

Emily Dickinson

Conteúdo

Lista de figuras

Lista de tabelas

Lista de algoritmos

1
INTRODUÇÃO

Este primeiro capítulo, introdutório, estabelece, nas secções 1.1 e 1.2, respectivamente, o âmbito e os objectivos do trabalho de dissertação realizado, e apresenta, na secção 1.3, a forma como a dissertação está organizada.

1.1 Âmbito

Uma atenção crescente tem sido dedicada, em anos recentes, à optimização em contextos multiobjectivo. Como parte desse interesse crescente, as meta-heurísticas têm vindo a ser adaptadas por forma a lidar com objectivos múltiplos, num esforço certamente motivado pelo sucesso da sua aplicação em contextos de objectivo único.

Algumas das principais tendências na investigação actual em meta-heurísticas podem ser perspectivadas como sendo baseadas na *flexibilização de meta-heurísticas*, entendendo-se por *flexibilização* a introdução de mecanismos de modificação dos seus componentes e estratégias elementares.

Igualmente resultado de significativa atenção recente, várias abordagens orientadas por objectos têm sido propostas para a área das meta-heurísticas de objectivo único, sobretudo com os objectivos de aproximar teoria e aplicação e incrementar a eficiência na implementação e comparação de métodos.

No entanto, até ao momento, e segundo o melhor conhecimento do autor, nenhuma proposta de abordagem surgiu para a área das meta-heurísticas multiobjectivo e, por outro lado, na literatura não é suficientemente enfatizado o facto de que as abordagens orientadas por objectos se apresentam particularmente apropriadas para a implementação e integração de estratégias genéricas de flexibilização.

1.2 Objectivos

O contexto descrito anteriormente sugere os principais objectivos para este trabalho:

- Propor e validar estratégias genéricas de flexibilização em meta-heurísticas multiobjectivo.
- Desenvolver um *framework* orientado por objectos que disponibilize uma arquitectura de base flexível para a construção, aplicação e comparação de meta-heurísticas multiobjectivo.

1.3 Estrutura da dissertação

Esta dissertação, elaborada em torno do trabalho desenvolvido em ordem à prossecução dos objectivos referidos, reflecte na sua estrutura o percurso realizado ao longo desse trabalho:

- O presente capítulo introduz o âmbito, os objectivos e a estrutura da dissertação.

- No capítulo 2 apresenta-se uma sistematização do domínio das meta-heurísticas, com excepção das meta-heurísticas multiobjectivo, identifica-se um conjunto importante de linhas de evolução na concepção deste tipo de algoritmos, que se designa por *flexibilização em meta-heurísticas*, e propõem-se algumas estratégias genéricas de flexibilização.
- O capítulo 3 compreende um desenvolvimento do esforço de sistematização do domínio das meta-heurísticas para a área particular das meta-heurísticas multiobjectivo, uma proposta de um *template* para pesquisa local multiobjectivo, e algumas propostas de estratégias genéricas de flexibilização para meta-heurísticas multiobjectivo baseadas em pesquisa local.
- No capítulo 4 apresenta-se uma sistematização dos principais conceitos em abordagens orientadas por objectos e do conjunto de abordagens orientadas por objectos para meta-heurísticas.
- O capítulo 5 é composto pela descrição do objectivo e pelo desenho detalhado do *framework*, que, no seu estado actual de desenvolvimento, foca, em particular, a área da pesquisa local multiobjectivo.
- No capítulo 6 é apresentado o problema abordado no caso de estudo - escalonamento multiobjectivo de tarefas numa máquina, minimizando o atraso máximo, o número de tarefas atrasadas e a soma ponderada dos atrasos - e a aplicação do *framework* desenvolvido a esse problema.
- O capítulo 7 apresenta a concepção e os resultados (e respectiva análise) do estudo computacional, realizado com os objectivos de estudar o comportamento das versões básicas de PSA e MOTS* sobre o problema considerado, conduzir uma avaliação preliminar do interesse da utilização das estratégias de flexibilização propostas e avaliar o potencial do *framework* desenvolvido na sua aplicação à comparação de configurações dos algoritmos.
- O capítulo 8 contém as conclusões finais da dissertação e aponta, em termos genéricos, algumas linhas de desenvolvimento futuro.

2
META-HEURÍSTICAS

Neste capítulo apresenta-se uma sistematização do domínio das meta-heurísticas, com excepção das meta-heurísticas multiobjectivo (objecto exclusivo do próximo capítulo), e com a qual não só se estabelece uma parte significativa do contexto no qual este trabalho se inscreve, mas também se concretiza uma análise do domínio, fundamental para o processo de desenvolvimento do *framework*.

É também a partir desta sistematização que se identifica um conjunto importante de linhas de evolução na concepção deste tipo de algoritmos, que se designa por *flexibilização em meta-heurísticas*. As propostas apresentadas afiguram-se particularmente interessantes para a aplicação de abordagens orientadas por objectos (esta verificação será desenvolvida no capítulo de apresentação do *framework*), o que reforça o interesse na sua consideração como um dos focos deste trabalho. Com este enquadramento, são propostas duas estratégias genéricas, utilizando vizinhanças variáveis e listas de candidatos, para meta-heurísticas baseadas em pesquisa local, em particular *Simulated Annealing* e Pesquisa Tabu.

Na secção 2.1 introduzem-se os principais conceitos de Optimização Combinatória, área em que se situam os problemas de optimização tratados com meta-heurísticas. A secção 2.2 é dedicada às heurísticas, tipo de procedimentos em que as meta-heurísticas se enquadram. Uma caracterização geral de meta-heurísticas é apresentada na secção 2.3, juntamente com uma descrição mais detalhada dos casos particulares considerados mais relevantes. A identificação das principais tendências de flexibilização é realizada na secção 2.4, na qual são ainda propostas duas estratégias genéricas integráveis em meta-heurísticas baseadas em pesquisa local. Por fim, é apresentado um pequeno conjunto de conclusões.

2.1 Optimização Combinatória

2.1.1 Introdução

Nos vários sectores de actividade surgem com frequência problemas de optimização que apresentam a particularidade de envolverem um número elevado mas finito de alternativas. Exemplos comuns são o escalonamento de operações em unidades fabris, o escalonamento de tripulações para prestação de serviços de transporte, a planificação de rotas de entregas, a localização de escolas e centros de prestação de cuidados de saúde ou a concepção de redes de telecomunicações.

Neste tipo de problemas é teoricamente possível enumerar todas as soluções e avaliar cada uma relativamente a um dado objectivo, previamente estabelecido. As que produzem o resultado mais favorável são consideradas óptimas. Contudo, do ponto de vista prático, é inviável a adopção desta estratégia para a resolução de problemas, uma vez que o número de soluções frequentemente cresce de forma incontrolável, com o tamanho do problema.

Ao longo das últimas cinco décadas tem sido realizado um vasto trabalho de investigação no sentido de desenvolver métodos de pesquisa óptimos que não exijam a análise explícita de cada alternativa. Com este trabalho de investigação estabeleceu-se, e tem vindo a evoluir, o campo da *Optimização Combinatória* (OC) e, conjuntamente, uma capacidade de resolver problemas do mundo real de dimensões crescentes.

Este apontamento introdutório, baseado em [Feo, Resende 1995] apresenta, de forma concisa, os aspectos mais relevantes do contexto em que surge a OC, bem como o seu objecto: os *Problemas de Optimização Combinatória* (POC).

2.1.2 Problemas de Optimização Combinatória

Um POC é um problema de optimização matemática com um conjunto de soluções admissíveis discreto. Na sua forma mais geral, pode ser enunciado do seguinte modo: dado um conjunto discreto S e uma função $f : S \to \Re$, encontrar um elemento $s^* \in S$ para o qual

$$f(s^*) = \min \{ f(s) \mid s \in S \}. \tag{2.1}$$

Os elementos do conjunto S são denominados *soluções admissíveis*, enquanto o conjunto em si é designado *espaço de soluções* ou *espaço de decisão*. À função f chama-se *função objectivo*. Sem perda de generalidade, restringir-se-á a presente discussão a problemas de minimização.

Tipicamente, um POC não será formulado de forma tão geral. Antes, será disponibilizada uma caracterização não enumerativa do espaço de soluções e uma forma algorítmica de avaliar o valor da função objectivo para cada solução admissível [Thienel 1995].

Veja-se, como exemplo desta última, o caso particular de um POC com uma função objectivo linear, dada por coeficientes de peso relativos aos elementos de um conjunto base, que compõem as soluções do problema. Está-se, neste caso, perante um *Problema de Optimização Combinatória Linear*, que, na sua forma mais geral, pode ser enunciado do seguinte modo: dado um conjunto discreto E de "entidades fundamentais", um conjunto $S \subseteq 2^E$ de subconjuntos de E, uma função $c : E \to \Re$ e sendo, para cada conjunto $F \subseteq E$, $c(F) = \sum_{e \in F} c(e)$, encontrar um conjunto $s^* \in S$ para o qual

$$c(s^*) = \min\{c(s) \mid s \in S\}. \tag{2.2}$$

É frequente e, num certo sentido, natural formular um POC como um *Problema de Optimização Inteira*, no qual as soluções $s \in S$ são descritas por vectores de variáveis inteiras (tipicamente binárias) e S é descrito por um conjunto de restrições de igualdade e/ou desigualdade. Esta forma de descrição constitui, ela própria, um exemplo de caracterização não enumerativa do espaço de soluções.

2.1.3 Complexidade Computacional

Em princípio, um POC poderia ser resolvido por uma enumeração exaustiva do espaço de soluções, com o cálculo do valor da função objectivo para cada solução admissível. No entanto, o facto, já apontado, de que o número de soluções admissíveis pode crescer de forma incontrolável com a dimensão do problema torna o método de enumeração impraticável na generalidade das situações. Com efeito, existem POC para os quais apenas são conhecidos algoritmos de resolução cujo tempo de cálculo aumenta de forma incontrolável (no sentido que se precisará de seguida) com o tamanho do problema (traduzindo, assim, o seu carácter combinatório "explosivo").

A teoria da *Complexidade Computacional* (ver, por exemplo, [Papadimitriou, Steiglitz 1982] ou [Nemhauser, Wolsey 1989]) disponibiliza um enquadramento adequado para o conjunto de questões associadas à "eficiência" dos algoritmos. Faz-se, aqui, apenas uma breve introdução às ideias fundamentais desta teoria e às suas implicações práticas, com base nas referências atrás indicadas.

A medida de desempenho (em termos de eficiência) mais utilizada para algoritmos baseia-se no tempo dispendido até à determinação da solução final (óptima). Para a tornar independente de particularidades de computadores, como a velocidade ou o conjunto de instruções, ou de linguagens de programação, é frequente esta medida ser referida ao número de passos algorítmicos elementares (operações aritméticas, comparações, etc.) de execução do algoritmo num computador hipotético.

Este número de passos varia, em geral, com a entrada (*"input"*) do algoritmo, ou seja, com a *instância* do problema que se pretende resolver. Uma *instância* (ou seja, uma concretização numérica) de um POC consistirá habitualmente de um *objecto combinatório* como um grafo, um conjunto de valores inteiros (organizados em vectores ou matrizes) ou outros. Para tratamento por computador, este objecto deverá ser codificado, ou representado, como uma sequência de símbolos sobre um alfabeto pré-definido (de que bits e ASCII constituem exemplos). O *tamanho* da entrada de um algoritmo, logo, de uma instância, será o comprimento dessa sequência de símbolos, ou seja, o número de símbolos que a compõem.

A *complexidade de um algoritmo*, para um tamanho de instância determinado, é o desempenho mais desfavorável do algoritmo (*"worst-case"*), para uma qualquer instância desse tamanho, e que, em geral, será função desse tamanho. São considerados particularmente úteis os algoritmos *polinomiais*, ou seja, aqueles cuja complexidade é uma função polinomial do tamanho da instância. Apresentam igual interesse os algoritmos cuja complexidade, não sendo polinomial, seja limitada polinomialmente. Os algoritmos cuja complexidade não é limitada polinomialmente são designados *exponenciais.*

Apenas são conhecidos algoritmos exponenciais para a resolução dos POC pertencentes à classe dos problemas *NP-difíceis* (*"NP-hard"*). Para a introdução deste conceito é importante estabelecer em primeiro lugar a relação entre *Problemas de Optimização* (PO) e *Problemas de Decisão* (PD). Um PD é um problema que exige uma resposta exclusivamente do tipo *sim-ou-não.* O PD correspondente à forma mais geral do POC, apresentada em (2.1), seria: para um determinado valor z, existe uma solução $s \in S$ tal que $f(s) \leq z$?

Existindo um algoritmo polinomial para resolver um PD, é possível resolver em tempo polinomial o PO correspondente, verificando-se igualmente o inverso. Como tal, os conceitos seguintes, introduzidos na área da Decisão, são de imediata aplicação na área da Optimização.

A classe P engloba os problemas para os quais se conhecem algoritmos que, para todas as instâncias do problema, produzem uma solução em tempo polinomial. A classe *NP* engloba os problemas para os quais é possível, em tempo polinomial, comprovar a veracidade de uma dada resposta. Verifica-se, portanto, que $P \subseteq NP$.

Um problema de decisão PD_1 é *transformável em tempo polinomial* num problema de decisão PD_2 se para qualquer instância *x* de PD_1 for possível construir em tempo polinomial uma instância *y* de PD_2, tal que a *y* corresponda uma resposta *sim* se e só se a *x* corresponder uma resposta *sim.*

Um PD diz-se *NP-completo* se pertence a NP e todos os problemas em NP podem nele ser transformados em tempo polinomial. Os problemas NP-completos apresentam algumas propriedades interessantes:

1. São problemas de elevada dificuldade computacional, não sendo conhecidos algoritmos polinomiais para nenhum deles.
2. Por outro lado, não se encontra provada a inexistência de algoritmos polinomiais para estes problemas.
3. A existir um algoritmo polinomial para um problema NP-completo, existiriam algoritmos polinomiais para todos os outros.
4. Inversamente, a demonstração de inexistência de um algoritmo polinomial para um dos problemas NP-completos, implicaria a inexistência de algoritmos polinomiais para todos os outros.

Um PD diz-se *NP-difícil* se todos os problemas em NP podem nele ser transformados em tempo polinomial, mas é desconhecido se esse problema pertence a NP. Os PO cujos PD correspondentes são NP-completos designam-se igualmente *NP-difíceis* (com efeito não pertencem a NP, uma vez que não são problemas de decisão). Muitos dos problemas práticos de interesse na área da Investigação Operacional, e em particular a grande maioria dos POC, são NP-difíceis [Lenstra et al. 1982], o que contribui decisivamente para a relevância da teoria da Complexidade Computacional nesta área.

Em termos práticos, poder-se-ia dizer que estes problemas, devido ao seu carácter combinatório, são intrinsecamente difíceis, e que os métodos de resolução não conseguem ultrapassar essa dificuldade. Esta perspectiva tem sentido, uma vez que, na prática, existem problemas difíceis cuja resolução não deverá naturalmente ser simples. Refira-se, no entanto, que para a avaliação prática da eficiência dos algoritmos, esta teoria é algo limitada, uma vez que se baseia numa análise do desempenho mais desfavorável (análise de *"worst-case"*), não tendo em conta que alguns algoritmos exponenciais podem apresentar um excelente desempenho em determinadas classes de instâncias (o que tem conduzido a outros tipos de análise, de carácter probabilístico).

A necessidade prática de resolver estes problemas tem levado a que, em alternativa aos *algoritmos exactos* - algoritmos que determinam a solução óptima - se tenham vindo a desenvolver outras abordagens de resolução deste tipo de problemas, que visam reduzir a carga computacional inerente à sua resolução. Estas abordagens podem apresentar diferentes características relativamente à qualidade da solução e ao tempo de resolução [Papadimitriou, Steiglitz 1982]:

1. *Algoritmos de aproximação*. Estes algoritmos produzem soluções que não são óptimas, mas relativamente às quais se possui a garantia de um limiar de afastamento em relação ao óptimo.

2. *Algoritmos probabilísticos*. Em alguns casos é possível desenvolver algoritmos que apresentam, em geral, um bom desempenho - aos níveis da qualidade da solução encontrada e do tempo consumido - assumindo determinadas distribuições de probabilidade para as instâncias do problema.

3. *Casos especiais*. Alguns casos particulares de problemas NP-difíceis podem ser de fácil resolução. Nestas situações, o facto de o problema geral ser NP-difícil é relativamente irrelevante.

4. *Algoritmos exponenciais*. Alguns destes algoritmos podem revelar-se de interesse prático, como pode ser o caso de algoritmos *pseudo-polinomiais* (algoritmos polinomiais em ordem ao tamanho da instância e ao maior inteiro da instância) ou algoritmos em média sub-exponenciais (não-polinomiais, mas com complexidade inferior a $2^{n^e}, e > 0$) mas que encontram sempre a solução óptima. A sua aplicação a instâncias de dimensão reduzida apresenta igualmente interesse prático.

5. *Pesquisa local*. Um dos métodos com maior sucesso na resolução de POC difíceis é o análogo discreto do procedimento de *"hill climbing"*, designado *pesquisa local*, que será tratado pormenorizadamente em parte posterior deste texto.

6. *Heurísticas*. Qualquer das abordagens anteriores, na ausência de uma garantia formal de desempenho, pode ser considerada uma heurística. Embora não garantindo a obtenção da solução óptima, as heurísticas produzem, de uma forma eficiente, soluções satisfatórias (soluções próximas do óptimo ou significativamente melhores do que as obtidas por métodos alternativos). Apesar de poderem ser, nalguns casos, pouco satisfatórias do ponto de vista matemático, estas abordagens são seguramente de interesse em situação práticas.

Uma outra abordagem, cuja aplicação à resolução de POC se tem revestido de algum sucesso prático, é a *Programação Lógica por Restrições* (*"Constraint Logic Programming"*). Esta abordagem surgiu na área da Inteligência Artificial como extensão, para resolução de POC, das linguagens de programação baseadas em lógica [Mackworth 1977]. O problema é modelado como um *Problema de Satisfação de Restrições*, através de linguagens declarativas para a descrição das restrições, utilizando uma variedade de formas algébricas e lógicas. As técnicas utilizadas visam reduzir a dimensão do espaço de pesquisa, aplicando restrições que condicionam a ordem em que se realiza a selecção das variáveis e a atribuição de valores a cada variável, numa pesquisa tipicamente em árvore. Em [Gervet 1998] são apontadas as principais tendências de evolução nesta área, com referências que disponibilizam um tratamento mais pormenorizado, não enquadrável no âmbito do presente texto.

2.2 Heurísticas

A obtenção de soluções admissíveis próximas (num sentido a precisar) do valor óptimo, e num tempo de cálculo razoável, é o objectivo de um conjunto de algoritmos, denominados *heurísticas*, que podem genericamente ser definidos da seguinte forma [Zanakis, Evans 1981]: procedimentos (geralmente) simples, frequentemente baseados no senso comum, que supostamente oferecerão uma solução boa (ainda que não necessariamente óptima) para problemas difíceis, de um modo fácil e rápido.

A utilização de heurísticas pode revelar-se interessante em vários tipos de situações [Diáz et al. 1996]:

- quando não existe um método exacto de resolução, ou os métodos disponíveis exigem demasiado tempo ou recursos computacionais;
- na eventualidade de não ser necessária a obtenção de uma solução óptima;
- em caso de reduzida fiabilidade dos dados do problema (não se justificando, assim, a procura do óptimo);
- perante limitações de tempo ou recursos computacionais;
- como passo intermédio na aplicação de outro algoritmo.

Sendo a redução dos requisitos de tempo de cálculo e recursos computacionais a vantagem principal das técnicas heurísticas em relação aos método exactos, existe, no entanto, um conjunto de vantagens adicionais que poderão ser relevantes na ponderação da sua utilização:

- permitem, em geral, uma maior flexibilidade na adaptação às características do problema;
- frequentemente possibilitam a determinação de mais do que uma solução, o que poderá permitir uma decisão sobre um leque de alternativas, eventualmente tendo em conta factores não quantificáveis não incluídos no modelo, ou uma lógica multicritério;
- baseando-se, em geral, em "regras de bom senso" ou inspirando-se mesmo em procedimentos tradicionais, a sua fundamentação é de mais simples entendimento pela generalidade dos decisores.

Um inconveniente importante da generalidade das heurísticas consiste em não ser possível garantir, à partida, um nível de qualidade da solução obtida. Para contornar tal inconveniente recorre-se com frequência a outro tipo de procedimentos. Por exemplo, a resolução de um problema mais simples, obtido por relaxação do problema original (isto é, ignorando algumas das suas restrições), permite obter um valor de referência para avaliar a qualidade da solução obtida pela heurística: sendo a solução óptima do problema relaxado "melhor ou igual" que a solução óptima do problema original, a proximidade da solução heurística à solução óptima do problema relaxado poderá constituir uma medida da sua qualidade.

Dependendo do modo como realizam a pesquisa e a construção das suas soluções, existem diversos tipos de heurísticas [Silver et al. 1980]:

1. *Métodos construtivos*. Consistem em acrescentar iterativamente componentes individuais à solução até se obter uma solução admissível. Os algoritmos *"greedy"*, construindo passo a passo a solução, e procurando o máximo benefício em cada passo, enquadram-se neste tipo de heurísticas.
2. *Métodos de decomposição*. O problema é dividido em subproblemas mais pequenos. O resultado de um constitui a informação de entrada do seguinte, e a resolução de todos proporciona uma solução para o problema global.
3. *Métodos de redução*. Procuram identificar alguma característica que a solução óptima deverá possuir, impondo a sua presença nas soluções candidatas, e simplificando assim o problema. Poder-se-á, por exemplo, detectar que algumas variáveis deverão tomar valores fixos, ou que estão correlacionadas.
4. *Manipulação do modelo*. Estas heurísticas modificam a estrutura do modelo, com o objectivo de simplificar a sua resolução, e deduzir a solução do problema original a partir da solução do problema modificado. A simplificação pode ser

conseguida por redução do espaço de soluções (linearizando funções não lineares, agrupando variáveis, etc.) ou, inclusive, aumentando-o (eliminando restrições).

5. *Métodos de pesquisa por vizinhanças.* Estes métodos partem de uma solução inicial admissível (obtida, por exemplo, por uma outra heurística) e, através de alterações dessa solução, vão passando de forma iterativa, e enquanto não se cumpra um determinado critério de paragem, a outras soluções admissíveis da sua vizinhança, guardando a melhor das soluções visitadas. Dentro desta categoria enquadra-se um número significativo de *meta-heurísticas*, objecto de análise neste trabalho.

2.3 Meta-heurísticas

O termo *meta-heurística* foi introduzido por Fred Glover [Glover 1986], designando uma estratégia mestra que guia e modifica outras heurísticas para produzir soluções além das que são normalmente geradas numa pesquisa de optimalidade local.

O assinalável sucesso destes métodos, que tem resultado num fluxo contínuo de publicações, deve-se a vários factores [Pirlot 1996]:

- referência a mecanismos de "optimização" da natureza, sendo as designações utilizadas e, até certo ponto, os princípios de algumas destas heurísticas originalmente inspirados por processos ou conceitos não relacionados com a optimização;
- aplicabilidade geral das abordagens;
- flexibilidade na consideração de restrições específicas em casos reais;
- excelentes compromissos entre qualidade da solução e facilidade de implementação ou tempo de cálculo.

Merecerá ainda destaque a robustez exibida por muitos destes métodos, com uma reduzida sensibilidade dos resultados obtidos a variações das características das instâncias do problema e à afinação de parâmetros.

A definição de Glover apresenta as meta-heurísticas como *"frameworks"* conceptuais genéricos para o desenvolvimento de heurísticas. As descrições de meta-heurísticas [Graccho, Porto 1999] assumem tipicamente a forma de *"templates"*, concretizados com a definição formal do problema e a especificação de um conjunto de estratégias e parâmetros. Estas características têm potenciado uma grande variedade de formas de aplicação de meta-heurísticas, para além da sua aplicação directa [Hansen 1998]:

- constituindo inspiração para o desenvolvimento de heurísticas especializadas para problemas específicos;
- permitindo a construção de métodos híbridos, por exemplo combinando componentes de diferentes meta-heurísticas;
- incorporando outras técnicas gerais da área da optimização;
- integrando-se com uma diversidade de técnicas implementacionais, aos níveis de estrutura e representação de dados, paralelização, afinação automática de parâmetros, etc.

Na sua forma tradicional, as meta-heurísticas raramente tratam o aspecto crucial na aplicação a um problema específico, que é a construção de uma *topologia* no espaço de soluções. No entanto, alguns princípios gerais segundo os quais se realiza esta construção estão consagrados nas meta-heurísticas. Entre estes princípios, os dois fundamentais são os seguintes:

1. *Pesquisa local*, em que uma solução é repetidamente substituída por uma outra solução que pertence a uma vizinhança da primeira (o conceito de vizinhança será oportunamente apresentado de uma maneira formal).
2. *Recombinação*, em que duas ou mais soluções se combinam através de algum mecanismo gerador de soluções, de forma a que a nova solução herde características das soluções geradoras.

As meta-heurísticas mais divulgadas na literatura integram um ou ambos os princípios. A adopção destes princípios como enquadramento conceptual de base permitirá apresentar uma descrição algorítmica de meta-heurísticas bastante abrangente (Figura 2.1): são métodos que partem de uma ou mais soluções iniciais admissíveis e, através de transformações dessas soluções, vão passando de forma iterativa, e enquanto não se cumpra um determinado critério de paragem, a outras soluções admissíveis, da sua vizinhança ou obtidas por recombinação, considerando-se como "óptima" a melhor das soluções visitadas.

Naturalmente que neste domínio, face aos inúmeros desenvolvimentos recentes, existe uma enorme tentação de realizar um esforço de sistematização que permita situar, num quadro unificado, as diferentes meta-heurísticas. No entanto, uma classificação rigorosa do domínio [Glover, Laguna 1997] é uma tarefa difícil e arriscada, uma vez que os defensores de métodos alternativos frequentemente divergem acerca da natureza essencial desses métodos. Em [Hansen 1998] é apresentada uma distinção básica, que assenta nos seguinte aspectos:

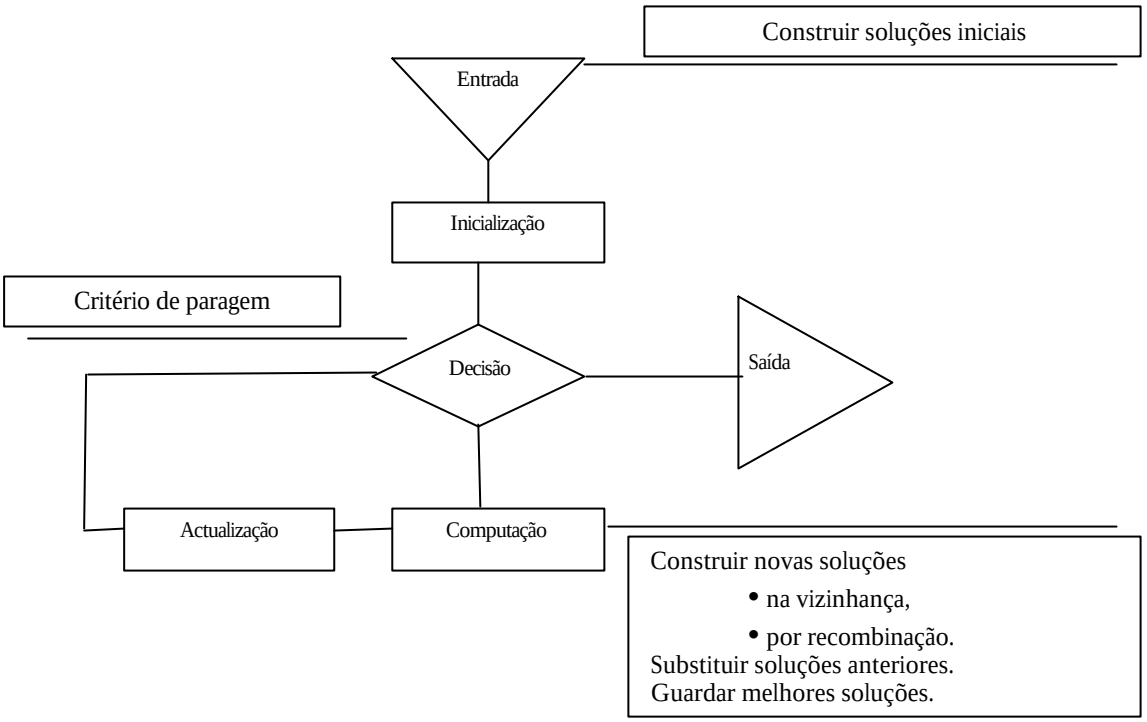


Figura 2.1 Meta-heurística - descrição algorítmica genérica

1. procedimento usado para a transformação das soluções: pesquisa local contínua (PLC), ou com rearranque/multiarranque[1] (PLR/M), ou recombinação (R);
2. número de soluções passadas de uma iteração para a seguinte: solução única (1), ou abordagens baseadas em populações (n).

Seguindo esta estrutura de classificação, o autor referido apresenta uma lista de meta-heurísticas (Tabela 2.1). A essa lista são aqui acrescentadas duas meta-heurísticas frequentemente referidas na literatura e com algum sucesso prático: *Noising Method* e *Ant Colony Optimization.*

As primeiras quatro meta-heurísticas (1 a 4) podem considerar-se como "clássicas", existindo, apesar de tudo, para as três seguintes (5 a 7) um número já relevante de publicações. Os quatro casos seguintes (8 a 11) são bastante recentes e têm apresentado alguns resultados merecedores de atenção. Como referido, sobre as duas últimas, têm surgido com alguma regularidade publicações ilustrativas do seu interesse.

[1] De uma forma sucinta, poder-se-á definir rearranque e multiarranque como a repetição da execução de um algoritmo a partir, respectivamente, da solução final obtida na execução anterior, ou de várias soluções iniciais diferentes.

	Meta-heurísticas			Referências
1.	Algoritmos Genéticos	R	n	[Holland 1975, Goldberg 1989]
2.	*Simulated Annealing*	PLC	1	[Kirkpatick et al. 1983, Vidal 1993]
3.	Pesquisa Tabu	PLC	1	[Hansen 1986, Glover 1986, Glover 1989, Glover 1990]
4.	Redes Neuronais	R	1	[Hopfield, Tank 1985]
5.	*GRASP*	PLR/M	n	[Feo, Resende 1989, Feo, Resende 1995]
6.	*Threshold Algorithms*	PLC	1	[Dueck, Scheuer 1990, Glover 1995b]
7.	*Scatter Search*	R	n	[Glover 1995a]
8.	*Variable Neighborhood Search*	PLC	1	[Mladenovic, Hansen 1997]
9.	*Cooperative Search Systems*	R	n	[Colorni et al. 1996, Toulouse et al. 1997]
10.	*Heuristic Concentration*	R	n	[Rosing, ReVelle 1997]
11.	*Jump Search*	PLR/M	1	[Tsumakitani, Evans 1998]
12.	*Noising Method*	PLC	1	[Charon, Hudry 1993]
13.	*Ant Colony Optimization*	R	n	[Colorni et al. 1991]

Tabela 2.1 Caracterização das principais meta-heurísticas e algumas referências

2.3.1 Meta-heurísticas baseadas em pesquisa local

As abordagens de *pesquisa local* remontam ao final da década de 1950, quando Bock [Bock 1958] e Croes [Croes 1958] desenvolveram os procedimentos de *"link exchange"* para o Problema do Caixeiro Viajante (*"Travelling Salesman Problem"*). Desde então, uma grande diversidade de algoritmos de *pesquisa local* têm sido propostos, procurando diferentes formas de evitar a optimalidade local.

Uma revisão da área pode ser obtida em [Yannakakis 1990] e um tratamento mais detalhado em [Aarts, Lenstra 1995]. Para uma sistematização dos conceitos fundamentais de vizinhanças refere-se [Papadimitriou, Steiglitz 1982].

O conceito de vizinhança

A questão central da *pesquisa local* é como realizar a passagem de uma solução para outra solução. Para a discussão deste tópico, é essencial começar por definir o que é a *vizinhança* $N(s)$ da solução s, ou seja, o conjunto de soluções "semelhantes" a s. O significado de "semelhante" deve aqui ser associado à possibilidade de obter uma solução $s' \in N(s)$ a partir de s, realizando apenas uma operação elementar, chamada *movimento*, sobre s (eliminar ou acrescentar um componente à solução, trocar elementos numa permutação, etc.).

Uma *vizinhança* N é um mapeamento $N : S \rightarrow 2^S$, em que 2^S denota o conjunto de todos os subconjuntos de S. Para cada $s \in S$, o conjunto $N(s)$ contém todas as soluções que podem ser atingidas com um movimento a partir de s. Estas soluções são designadas *vizinhos* de s.

Uma estrutura de vizinhança N pode ser representada por um *grafo de vizinhança* $G = (V, A)$, no qual $V = S$ e $(s, s') \in A$ sse $s' \in N(s)$. Se para todo o $(s, s') \in A$ se verificar simultaneamente $(s', s) \in A$, a vizinhança é *simétrica* e pode ser representada por um grafo não orientado.

Uma vizinhança é frequentemente definida por um *conjunto de operadores* OP. Um operador é um mapeamento $op : S^{op} \rightarrow S$ com domínio $S^{op} \subseteq S$. O conjunto de vizinhos $N(s)$ para uma solução $s \in S$ é então dado por

$$N(s) = \{op(s) \mid op \in OP, s \in S^{op}\}. \tag{2.3}$$

Uma solução $s \in S$ é um *óptimo local* em relação a N se $f(s) \leq f(s')$, para todo o $s' \in N(s)$. Uma vizinhança N em que todos os óptimos locais em relação a N são também óptimos globais designa-se *vizinhança exacta*.

Pesquisa local simples ("Descent" ou "Hill Climbing")

Os métodos de pesquisa local baseiam-se em determinar, de entre os elementos da vizinhança $N(s)$ da solução actual s, um que tenha um melhor valor de acordo com algum critério pré-definido, passando a considerar esse elemento como a nova solução actual, e repetindo a operação até que se considere que não é possível encontrar uma solução melhor, ou porque não exista nenhum elemento melhor na vizinhança dessa solução, ou porque se verifique algum critério de paragem.

No algoritmo de pesquisa local mais simples, chamado *Iterative Improvement* (Algoritmo 2.1), a vizinhança da solução actual é percorrida até ser encontrada uma melhor solução; se tal suceder a solução encontrada torna-se a nova (e melhor) solução actual; caso contrário, o algoritmo termina.

Algoritmo *Iterative Improvement*

Calcular uma solução de partida admissível $s \in S$;

Enquanto existir uma solução melhor em $N(s)$

Determinar um $s' \in N(s)$ com $f(s') < f(s)$;

$s = s'$;

Algoritmo 2.1 *Iterative Improvement*

Numa variante deste esquema, conhecida como *Steepest Descent* (Algoritmo 2.2), em alternativa a seleccionar qualquer solução melhor do que a actual, selecciona-se a melhor solução da vizinhança.

Algoritmo *Steepest Descent*

Calcular uma solução de partida admissível $s \in S$;

Enquanto existir uma solução melhor em $N(s)$

Determinar $s' \in N(s)$ com $f(s') \leq f(s'')$, para todo o $s'' \in N(s)$;

$s = s'$;

Algoritmo 2.2 *Steepest Descent*

Opções gerais

A aplicação de meta-heurísticas baseadas em pesquisa local exige um conjunto de opções de carácter estratégico, que influenciam de forma determinante o seu desempenho. Algumas dessas opções são comuns à generalidade das meta-heurísticas [Pirlot 1996]:

- *Escolha de um critério de paragem*. Contrariamente à pesquisa local simples, a generalidade das meta-heurísticas baseadas em pesquisa local não tem implícito um critério de paragem. As variantes genéricas mais comuns consistem em terminar a execução do algoritmo após um número fixo de iterações, após um determinado número de iterações sucessivas sem alteração da melhor solução, ou quando é atingido um tempo limite de execução.
- *Selecção do espaço de soluções, função objectivo e estrutura de vizinhança*. Desejavelmente, a movimentação de solução para solução vizinha, através do espaço de soluções, deverá ser tão fácil quanto possível, por motivos de eficiência computacional.

É prática habitual ter em conta restrições do problema original, através da inclusão na função objectivo de penalizações associadas à sua violação. Assim, revela-se de grande importância prática a decisão de quais as restrições rígidas, que deverão ser consideradas na definição do espaço de soluções e de quais as que poderão ser relaxadas (isto é, tratadas como indicado).

Refira-se ainda que uma estrutura de vizinhança deverá sempre garantir que a partir de uma qualquer solução se possa atingir qualquer outra, através de uma sequência de movimentos.

- *Escolha da solução inicial*. A solução de partida para o processo de pesquisa pode ter uma grande influência sobre a possibilidade de mais rapidamente se cair ou não num óptimo local. Em geral, será interessante que o desempenho do algoritmo seja independente da solução de partida.

  Se, por um lado, uma boa solução de partida pode confinar a pesquisa a uma região da qual é difícil escapar, por outro, em situações em que as boas soluções tenham uma estrutura particular, dificilmente alcançável com movimentos elementares a partir de uma solução inicial aleatória, poderá fazer sentido utilizar soluções de partida cuidadosamente escolhidas.

Simulated Annealing

As meta-heurísticas baseadas em pesquisa local estabelecidas como "clássicas" na literatura são o *Simulated Annealing* (SA) e a *Pesquisa Tabu* (TS, do inglês *Tabu Search*). Ambas as meta-heurísticas, nas suas versões mais elementares, se baseiam em pesquisa local contínua. Uma excelente introdução a estas meta-heurísticas, na qual se baseia a seguinte apresentação, pode ser encontrada em [Pirlot 1996].

Na versão base (tradicional) do SA [Kirkpatick et al. 1983] (Algoritmo 2.3) selecciona-se aleatoriamente uma solução na vizinhança da solução actual, que será aceite como nova solução actual se for uma melhor solução. Caso contrário, será aceite com uma probabilidade que será dada por uma função decrescente com a extensão da deterioração, e crescente com um parâmetro designado *temperatura*. Este último parâmetro é gradualmente reduzido de forma a tornar o algoritmo mais selectivo na aceitação de novas soluções.

Algoritmo *Simulated Annealing*

Calcular uma solução de partida admissível $s \in S$;

Inicializar a melhor solução encontrada $s^* = s$;

Inicializar a temperatura T;

Enquanto não se cumprir o critério de paragem

Seleccionar aleatoriamente $s' \in N(s)$;

Se $f(s') < f(s)$

$s = s'$;

Se $f(s) < f(s^*)$

$s^* = s$;

Se não

Seleccionar aleatoriamente um número $p \in [0,1]$;

Se $p \leq e^{-\frac{f(s')-f(s)}{T}}$

$s = s'$;

Actualizar a temperatura T;

Algoritmo 2.3 *Simulated Annealing*

Além do conjunto de decisões genéricas, já referidas, algumas decisões específicas ao SA envolvem:

1. *Escolha da probabilidade de aceitação.* A probabilidade utilizada no algoritmo apresentado corresponde a uma distribuição de Boltzmann, numa analogia à termodinâmica. Não parecem existir, actualmente, razões especiais para privilegiar esta opção, tradicionalmente utilizada na prática.

2. *Escolha de um esquema de arrefecimento ("cooling schedule").* O esquema mais comum consiste em partir de uma temperatura inicial T_0, manter a temperatura durante um número constante de passos L (*plateau*), e após cada série de L passos decrescê-la (geometricamente) através do produto por um factor fixo a, $0 < a < 1$. Ou seja, após a série de passos k, a temperatura será $T_k = T_0 \cdot a^k$.

3. *Escolha de critério de paragem específico.* Variantes particulares ao SA podem consistir em ser atingida uma temperatura mínima, ou o número de movimentos aceites

num determinado número de iterações sucessivas ser inferior a um limiar estabelecido em função dos parâmetros específicos do algoritmo.

Uma meta-heurística que apresenta fortes semelhanças com o SA é o *Threshold Accepting* [Dueck, Scheuer 1990]. Enquanto o SA recorre a uma regra de aceitação probabilística para soluções que provocam uma deterioração do valor da função objectivo, o *Threshold Accepting* utiliza uma regra de aceitação determinística. Um movimento será aceite se não deteriorar o valor da função objectivo em mais do que um limiar V que, à semelhança do SA, é decrescido geometricamente ao longo da execução do algoritmo, ou seja, $V_k = V_{k-1} \cdot \alpha$ e $V_k = V_0 \cdot \alpha^k$, com $0 < \alpha < 1$.

Pesquisa Tabu

A *Pesquisa Tabu* (TS) [Glover 1986] é uma técnica de pesquisa local orientada pela utilização de estruturas de memória adaptativas ou flexíveis. Na sua configuração mais básica (Algoritmo 2.4) utiliza uma lista, designada *lista tabu*, de *atributos* dos últimos movimentos realizados (em número previamente fixado, ou dinamicamente ajustado durante a execução). Esta lista é utilizada com o objectivo de evitar a ocorrência de movimentos inversos dos realizados recentemente, não permitindo por um lado o regresso a soluções anteriores e, sobretudo, garantindo diversificação da pesquisa, através da sua orientação para regiões do espaço de pesquisa ainda não exploradas e procurando, assim, evitar os óptimos locais. Nas versões básicas de TS, em alternativa à pesquisa numa vizinhança, é feita, com alguma frequência, uma pesquisa numa *sub-vizinhança* SN, isto é, num subconjunto da vizinhança, de dimensão definida e constituído por elementos retirados aleatoriamente da vizinhança.

Em cada iteração, a solução actual é substituída pela melhor solução encontrada na sua vizinhança, que não seja proibida pela lista tabu, ou que, caso o seja, conduza a uma solução *satisfatória*. Este carácter *satisfatório* é definido pela satisfação de um dado *critério de aspiração*, que será verificado, por exemplo, quando uma solução for melhor do que qualquer solução encontrada até ao momento, ou for melhor do que as soluções que originaram os atributos da lista que a fazem tabu.

Além do conjunto de decisões genéricas já apontadas, para a aplicação da TS é necessário decidir quais os atributos a guardar na lista, qual o comprimento máximo que esta deverá ter e qual o critério de aspiração. A utilização de outros tipos de estruturas de memória, de médio e longo prazo, é feita, normalmente, com objectivos de intensificar a exploração de regiões promissoras do espaço de soluções (*intensificação*) ou de diversificar a pesquisa, orientando-a para regiões inexploradas (*diversificação*).

Algoritmo Pesquisa Tabu

Calcular uma solução de partida admissível $s \in S$;

Inicializar a melhor solução encontrada $s^* = s$;

Inicializar a lista tabu $TL = \{\}$;

Enquanto não se cumprir o critério de paragem

Determinar $s' \in SN(s)$ com $f(s') \leq f(s'')$, para todo o $s'' \in SN(s)$, **e**

(TL não faz (s,s') tabu **ou** s satisfaz o critério de aspiração);

Se $f(s') < f(s)$

Inserir atributos de (s,s') em TL, removendo o primeiro

elemento se TL estiver completa;

Se $f(s') < f(s^*)$

$s^* = s'$;

$s = s'$;

Algoritmo 2.4 Pesquisa Tabu

GRASP

Na área da pesquisa local com multiarranque, a meta-heurística que tem vindo a assumir maior relevância é o *GRASP (Greedy Randomized Adaptive Search Procedure)* [Feo, Resende 1995] (Algoritmo 2.5).

Cada iteração de *GRASP* consiste de duas fases: construção e pesquisa local. Na primeira fase é construída uma solução admissível, acrescentando-se à solução um *componente* de cada vez, que é seleccionado de forma *"greedy"* de entre todos os elementos de uma *lista restrita de candidatos* (LRC). As soluções geradas pela fase construtiva do *GRASP* não serão garantidamente óptimos locais em relação a vizinhanças simples, pelo que será quase sempre benéfica a aplicação adicional de um método de pesquisa local, procurando melhorar a solução construída.

O carácter adaptativo do método resulta do facto de que o benefício associado a cada elemento é actualizado em cada iteração da fase de construção, reflectindo as alterações introduzidas pela selecção do elemento anterior. O carácter probabilístico é introduzido pela selecção aleatória de um dos melhores candidatos da LRC, não necessariamente o melhor.

Algoritmo *GRASP*

$f(s^*) = \infty$;

Enquanto não se cumprir o critério de paragem

$s = \{\}$;

Enquanto a construção de s não estiver completa

Construir lista restrita de candidatos *LRC*;

Seleccionar aleatoriamente um elemento c de *LRC*;

$s = s \cup \{c\}$;

Adaptar a função *"greedy"*;

Realizar pesquisa local, com solução inicial s e solução

obtida s';

Se $f(s') < f(s^*)$

$s^* = s'$;

Algoritmo 2.5 GRASP

Novamente, são aqui inteiramente pertinentes algumas opções genéricas, nomeadamente em relação ao critério de paragem e à selecção apropriada de função objectivo, espaço de soluções e estrutura de vizinhança. A escolha da solução inicial é, neste caso, tratada explicitamente pelo algoritmo, sendo necessário definir e afinar apenas um parâmetro, relativo à composição da LRC:

- adoptando uma *restrição de valor*, seleccionam-se todos os candidatos com um valor que fique dentro de um limiar α(%) relativamente ao valor *"greedy"*;
- adoptando uma *restrição de cardinalidade*, seleccionam-se os β melhores candidatos.

2.3.2 Meta-heurísticas baseadas em recombinação

Nesta classe de meta-heurísticas, os mecanismos geradores de soluções, que garantem a herança de características das soluções geradoras, são de natureza bastante distinta entre si.

A área é dominada pelos *Algoritmos Genéticos* (GA) [Holland 1975, Goldberg 1989], inspirados pelo processo de evolução das espécies. Outras meta-heurísticas desta classe, inspiradas por processos da natureza são as *Redes Neuronais Artificiais* [Hopfield, Tank 1985], e

a *Ant Colony Optimization* [Colorni et al. 1991], inspirada pelo comportamento das formigas em busca de alimento.

A *Scatter Search* (SS) [Glover 1995a] apresenta afinidades com os GA. No entanto, as suas raízes situam-se numa área completamente distinta, a do desenvolvimento de estratégias para combinação de regras de decisão e restrições [Glover 1963], com o objectivo de obter um processo composto capaz de produzir melhores soluções que os elementos originais. Igualmente com uma origem diversa surgiu recentemente uma outra meta-heurística baseada em recombinação, a *Heuristic Concentration* (HC) [Rosing, ReVelle 1997], que procede em duas etapas: na primeira, forma um conjunto de soluções "sub-óptimas", cujos atributos tenham alguma probabilidade de estar presentes em soluções óptimas; posteriormente, a partir desse conjunto, constrói a melhor solução possível, combinando esses atributos.

No contexto do presente trabalho, será de interesse rever com algum pormenor os GA e a SS, que apresentam, conforme já referido, algumas afinidades.

Algoritmos Genéticos

Os Algoritmos Genéticos (GA, do inglês *Genetic Algorithms*) (Algoritmo 2.6) são algoritmos de optimização que se inspiram em certos aspectos da evolução genética de uma espécie. Os GA lidam com *populações* de soluções, que fazem *evoluir* através de operadores de *selecção*, *cruzamento* (*"crossover"*) e *mutação*.

Um GA simula este processo, tomando uma *população inicial* de *indivíduos* (soluções) e aplicando *operadores genéticos* em cada *reprodução*. Cada *indivíduo* tem um determinado *grau de aptidão* (ou *"fitness"*, frequentemente associado ao valor da função objectivo). Aos *indivíduos mais aptos* é dada a oportunidade de *reprodução* (geração de novas soluções) através da *troca de informação genética* (características das soluções), num processo de *cruzamento* (recombinação), com outros *indivíduos altamente aptos*. Desta forma são produzidas novas *gerações*, que *partilham características dos pais*. Após o *cruzamento*, é frequentemente aplicada uma *mutação*, alterando aleatoriamente alguns *genes*. A nova *geração* pode substituir totalmente a *população* anterior (abordagem *geracional*) ou apenas os *indivíduos menos aptos* (abordagem *estacionária*). Este ciclo *avaliação-selecção-reprodução* é repetido até se verificar um critério de paragem.

Algoritmo Algoritmo Genético

Construir uma população inicial P, com soluções $s \in S$;

Determinar em P qual a melhor solução s^*;

Enquanto não se cumprir o critério de paragem

Seleccionar, com reposição, M pares de indivíduos de P, sendo a probabilidade de selecção crescente com a qualidade do indivíduo;

Para cada par

Seleccionar aleatoriamente um número $p \in [0,1]$;

Se p < probabilidade de cruzamento χ

Gerar um par de filhos com operador de cruzamento;

Se não

Gerar um par de filhos idênticos aos pais;

Para cada filho s

Aplicar com probabilidade μ o operador de mutação;

Se $f(s) < f(s^*)$

$s^* = s$;

Seleccionar, sem reposição, 2M indivíduos de P, com probabilidade de selecção decrescente com a qualidade do indivíduo;

Substituir os 2M indivíduos pelos filhos;

Algoritmo 2.6 Algoritmo Genético

A aplicação de GA envolve um conjunto relativamente extenso de decisões [Pirlot 1996], constituindo preocupação fundamental a manutenção de um certo grau de diversidade na população, por forma a permitir explorar o máximo possível de regiões interessantes do espaço de soluções.

A dimensão da população deverá ser ponderada tendo em atenção, por um lado, a necessidade de manter diversidade e, por outro, a eficiência computacional. O operador de

selecção deverá ser concebido de forma a seleccionar os indivíduos mais aptos, mas sem conduzir a uma *convergência prematura* (situação em que todos os elementos são idênticos, ou seja, a população não apresenta diversidade). A taxa de substituição deverá obedecer a critérios semelhantes. Por último, o operador de mutação poderá desempenhar um papel preponderante, quer no sentido de evitar a convergência prematura, quer na diversificação da pesquisa.

O esquema de codificação das soluções assume aqui um papel crucial, já que o algoritmo processa representações específicas, recombinando-as. Por outro lado, a escolha dos operadores de cruzamento e selecção deve ser realizada de forma a permitir a progressiva proliferação, na população, de boas propriedades das soluções.

Scatter Search

Embora não directamente inspirada por processos presentes na natureza, a *Scatter Search* (SS) apresenta afinidades com os GA. Esta abordagem trabalha [Pirlot 1996] sobre um modelo espacial em vez de um modelo genético. *Soluções de referência* são combinadas, possivelmente em grupos de mais do que duas, através de *combinações estruturadas*, em vez de *cruzamentos genéticos*.

Os princípios fundamentais da SS são os seguintes [Glover et al. 2000]:

1. Um conjunto de *soluções elite*, adequadamente diversificadas, irá tipicamente conter informação útil sobre a forma ou a localização de soluções óptimas.
2. Na combinação de soluções, como forma de explorar essa informação, é importante dispor de mecanismos capazes de construir combinações para além das regiões em que se localizam as soluções geradoras. Assume, igualmente, importância a incorporação de heurísticas que permitam transformar as soluções combinadas em novas soluções, tornando-as admissíveis ou melhorando-as. O duplo propósito destes mecanismos de combinação é incorporar diversidade e qualidade nas soluções geradas.
3. O processamento de múltiplas soluções em simultâneo, como base para criar combinações, aumenta as oportunidades de exploração de informação contida na reunião de *soluções elite*.

Os mecanismos utilizáveis na SS não estão restringidos a um *design* único ou uniforme, permitindo, assim, a exploração de diversas possibilidades estratégicas. Em consequência, fará sentido propor, não um algoritmo concreto, mas um *template* genérico (Algoritmo 2.7), baseado nos princípios acima enunciados.

Template *Scatter Search*

Enquanto não se cumprir o critério de paragem

Gerar um conjunto de soluções I, garantindo um dado nível de diversidade;

Aplicar processos heurísticos para melhoria das soluções obtidas;

Seleccionar um subconjunto das melhores soluções como conjunto de referência R;

Fazer

Criar subconjuntos de R;

Para cada subconjunto

Criar novas soluções através de combinações estruturadas das soluções do subconjunto;

Modificar as soluções obtidas de forma a torná-las admissíveis;

Aplicar processos heurísticos para melhoria das soluções obtidas;

Extrair um conjunto das melhores soluções obtidas, segundo critérios de qualidade e diversidade, e actualizar R;

Até R não sofrer alterações;

Algoritmo 2.7 *Scatter Search*

Os mecanismos genéricos a instanciar numa aplicação específica da SS são, então:

1. *Método de geração diversificada*. Gera um conjunto de soluções diversificadas, colocando o ênfase na sua diversidade, mais do que na sua qualidade. Recorrerá tipicamente à aleatorização, controlada por mecanismos do tipo *memória de frequência*, que registam a frequência de ocorrência de atributos das soluções, enviesando o mecanismo gerador de soluções, no sentido de privilegiar a geração de soluções com os atributos menos frequentes.

2. *Método de melhoria*. Transforma uma solução em outra(s) melhorada(s). Um requisito fundamental deste mecanismo será a capacidade de lidar com soluções admissíveis e soluções não-admissíveis.

3. *Método de actualização do conjunto de referência*. Visa construir e actualizar o conjunto de referência, que deverá ser organizado de forma a permitir um acesso eficiente, incorporando soluções de acordo com critérios de qualidade e diversidade.

4. *Método de geração de subconjuntos*. Opera sobre o conjunto de referência, produzindo um ou mais tipos de subconjuntos das suas soluções, como base para criar soluções combinadas.

5. *Método de combinação de soluções*. Transforma um subconjunto de soluções numa ou mais soluções combinadas, recorrendo a *combinações estruturadas*.

Numa perspectiva de modelo espacial, o processo de gerar combinações estruturadas de um conjunto de soluções de referência pode ser visto [Glover 1998a] como a geração de *caminhos entre, e para além,* dessas soluções. Soluções desses caminhos serão novamente utilizadas para a geração de caminhos adicionais.

Uma estratégia associada à TS, mas com fortes afinidades com a SS, designada *Path Relinking* (PR), baseia-se na geração de soluções ao longo destes caminhos [Glover 1998b]. Numa referência muito breve, poder-se-á descrever esta estratégia como consistindo da realização de sucessivos movimentos sobre uma *solução inicial*, de forma a progressivamente a tornar idêntica a uma *solução orientadora*. Os mecanismos para a realização desses movimentos são fortemente dependentes da representação da solução. Em geral, actuarão introduzindo na *solução inicial* atributos da *solução orientadora*, ou modificarão os atributos da *solução inicial* de forma a reduzir uma medida adequada de distância entre as duas soluções, no espaço das soluções.

Algumas variantes baseiam-se na inversão dos papéis das soluções ou na simultaneidade de papéis, com ambas as soluções a assumirem os papéis de *solução inicial* e *solução orientadora*.

2.4 Flexibilização em meta-heurísticas

2.4.1 Introdução

O interesse no conceito de *flexibilização em meta-heurísticas* está naturalmente ligado aos objectivos fundamentais normalmente associados aos algoritmos deste tipo, que são, como é reconhecido em [Feo, Resende 1995], encontrar uma solução óptima ou quase-óptima, e chegar a essa solução com um esforço computacional mínimo.

Por *flexibilização* entende-se a introdução de mecanismos de modificação dos componentes e das estratégias elementares de pesquisa local e de recombinação.

Componentes e estratégias elementares

A identificação de componentes e de estratégias elementares, na pesquisa local e na recombinação, depara-se com dificuldades idênticas às da realização de uma classificação rigorosa do domínio das meta-heurísticas, que resultam da diversidade de perspectivas relativamente à natureza essencial dos diversos métodos.

Em [Glover, Laguna 1997] a evolução das meta-heurísticas é vista como baseada numa variedade de interpretações do que constitui pesquisa "inteligente". Estas interpretações motivam escolhas conceptuais, que podem ser usadas para fins de classificação. Entre os aspectos básicos que são alvo dessas escolhas salientam-se:

- a utilização de memória adaptativa (algoritmos com memória, sem memória);
- o tipo de exploração de vizinhança utilizado (pesquisa sistematizada para seleccionar o próximo movimento ou melhorar a solução actual, ou métodos baseados em amostragem aleatória);
- o número de soluções passadas de uma iteração para a seguinte (métodos com uma solução actual única, abordagens baseadas em populações).

Adicionalmente, as meta-heurísticas têm vindo a incorporar frequentemente outras estratégias, por forma a orientar o processo de pesquisa:

- modificação estratégica da função de avaliação (da qual o *Simulated Annealing* constitui um exemplo);
- modificação estratégica da vizinhança, dela excluindo, ou nela introduzindo, algumas soluções;
- procedimentos de rearranque aleatório simples, ou perturbação aleatória;
- utilização de soluções parciais, fugindo à utilização exclusiva de movimentos de "transição" sobre soluções completas.

Em [Feo, Resende 1995] consideram-se as seguintes categorias de conceitos heurísticos fundamentais:

- construção de soluções,
- perturbação de soluções,

- repetição de procedimentos e critérios de rearranque,
- decomposição ou condicionamento do problema e
- critérios de paragem.

Para cada uma destas categorias, têm sido concebidos uma série de mecanismos algorítmicos que, por sua vez, têm sido combinados com sucesso em técnicas híbridas.

Em [Vaessens et al. 1995], um esforço de unificação e classificação é conduzido sobre um *template* de pesquisa local. A unificação entre pesquisa local e recombinação é conseguida através da consideração de um conjunto de soluções actuais (constituindo o caso de uma solução actual única, um caso particular) e considerando o processo de recombinação como definindo uma estrutura de *hiper-vizinhança*, associada a um *cluster* de soluções. Incorporando estas alterações, o *template* é construído como *generalização* do algoritmo *Iterative Improvement*.

É possível observar nestes casos a existência de elementos comuns às várias perspectivas, bem como a possibilidade de reescrita de cada perspectiva segundo as restantes. Assim, e devido à afinidade que o esquema de *generalização* do *Iterative Improvement* apresenta com o conceito de *flexibilização*, optou-se, neste trabalho, pela utilização dessa perspectiva para identificação dos componentes e estratégias mais elementares, mantendo, no entanto, e em coerência com o que tem sido feito ao longo do presente texto, a distinção entre pesquisa local e recombinação.

No âmbito do Problema de Optimização Combinatória, os componentes básicos identificados são: a solução, os critérios de admissibilidade da solução, a função objectivo e os dados do problema.

Na pesquisa local, os componentes básicos são: a solução inicial, a solução actual, a melhor solução encontrada, as soluções vizinhas e correspondentes movimentos e a estrutura de vizinhança. As estratégias que articulam estes componentes são as seguintes: pesquisa de soluções na vizinhança, substituição da solução actual, repetição de procedimentos e paragem.

Na recombinação consideram-se os seguintes componentes: população inicial, população actual, melhor solução encontrada, conjuntos de soluções geradoras, soluções geradas e método de recombinação. Estes componentes são articulados pelas seguintes estratégias: formação de conjuntos de soluções, substituição de soluções, repetição dos procedimentos e paragem.

Tendências de flexibilização em meta-heurísticas

As estratégias de *flexibilização* consideradas no presente trabalho enquadram-se nalgumas das principais tendências na investigação actual em meta-heurísticas:

1. criação de novas formas de fuga à optimalidade local, quer no domínio da pesquisa local, quer no domínio da recombinação;
2. generalização de conceitos fundamentais particulares a determinadas meta-heurísticas de pesquisa local ou recombinação;
3. utilização conjunta de conceitos provenientes de meta-heurísticas distintas, ou mesmo, a um nível mais alto, utilização conjunta de várias meta-heurísticas.

A primeira destas tendências vem, naturalmente, desde há bastantes anos, assistindo-se ainda hoje em dia à génese de novas meta-heurísticas, enquanto as restantes tendências se têm vindo a manifestar recentemente com crescente intensidade.

No âmbito da primeira tendência apontada, o enquadramento proporcionado pelo conceito de *flexibilização* pode abranger as próprias meta-heurísticas clássicas:

- O SA recorre a uma estratégia de *selecção aleatória de uma solução* na vizinhança e a um *critério elaborado de aceitação* de soluções, que permite a aceitação de soluções de *qualidade inferior* à da solução actual.
- A TS faz uso de uma *restrição da vizinhança*, que permite afastar a pesquisa de óptimos locais, em combinação com uma alteração do *critério de aceitação*, de forma a aceitar soluções de *qualidade inferior* à da solução actual.

Também algumas meta-heurísticas mais recentes se enquadram nesta tendência:

- Os *Noisy Methods* [Charon, Hudry 1999] são uma família de métodos que trabalham com base em *perturbações* da própria instância do problema (acrescentando "ruído" à *função objectivo*) e *execuções repetidas* de estratégias elementares de pesquisa local. Uma outra evidência de que os diversos princípios gerais associados às meta-heurísticas se podem reescrever em diversas perspectivas é o facto de que o SA e o *Threshold Accepting* podem, de certa forma, ser encarados como casos particulares dos *Noisy Methods*.
- A *alteração dinâmica da estrutura de vizinhança* está na base da *Variable Neighborhood Search* (VNS) [Mladenovic, Hansen 1997], que será descrita em pormenor posteriormente.

Ao nível dos procedimentos de construção de soluções iniciais, em particular dos algoritmos *"greedy"*, a base do GRASP é a utilização de um *grau de "greediness"*, com *selecção aleatória* de entre os componentes candidatos, em alternativa à construção *"greedy"* pura.

No quadro da segunda tendência referida, diversos conceitos têm vindo a autonomizar-se e a ser implementados como estratégias gerais:

- A utilização de *diferentes soluções iniciais* constitui a base das *estratégias de rearranque e multiarranque*.
- O elemento diferenciador entre *Iterative Improvement* e *Steepest Descent*, está na base de duas estratégias gerais de pesquisa de vizinhança: *"first improvement"* ou *"best improvement"*.
- Diversas transformações genéricas de estruturas de vizinhança surgiram da área da TS: a consideração de uma amostra aleatória da vizinhança, nas *sub-vizinhanças*; a selecção de *listas de candidatos* de movimentos potencialmente interessantes, considerados em posteriores iterações; *vizinhanças de profundidade variável*, em que para cada solução da vizinhança se constrói a respectiva vizinhança, e assim sucessivamente até se atingir o nível de profundidade pretendido.
- A evolução dos Algoritmos Genéticos e da *Scatter Search* tem sido orientada no sentido de estas meta-heurísticas serem vistas como *templates* de algoritmos baseados em recombinação, sobre os quais é possível implementar, de várias formas, diferentes mecanismos, como a constituição de subconjuntos de soluções geradoras, a recombinação ou a gestão da composição da população.

A terceira tendência tem-se manifestado através de uma intensificação dos esforços de investigação em abordagens de:

- *hibridização*, em que se procura que diferentes meta-heurísticas se complementem, tirando partido das suas qualidades específicas;
- *paralelização*, em que versões paralelas das diversas meta-heurísticas são propostas, com o objectivo de tratar problemas de dimensões realistas em tempos computacionais aceitáveis.

No contexto deste trabalho, as tendências identificadas têm os seguintes reflexos:

- Procura-se identificar "tema e variações" nas meta-heurísticas baseadas em pesquisa local, com o objectivo de vir a tirar partido de infra-estruturas comuns que, designadamente, favoreçam a utilização de estratégias gerais, bem como a integração em abordagens de hibridização e paralelização. Esta identificação foi já parcialmente realizada neste capítulo e será completada nos capítulos seguintes

com a proposta de um *template* para pesquisa local multiobjectivo e uma apresentação geral do *template* de [Vaessens et al. 1995].

- Utilizam-se estratégias que evoluíram no sentido da generalização, destacando dois casos que na literatura têm sido referidos como particularmente promissores: as *vizinhanças variáveis* que, constituindo o cerne da *Variable Neighborhood Search* (VNS), apresentam um forte potencial de aplicação no seio de outras meta-heurísticas; e as *listas de candidatos*, oriundas da área da TS.
- Procura-se tirar partido de abordagens de *hibridização* e *paralelização*, nomeadamente no caso de meta-heurísticas cuja estrutura, à partida, potencie a respectiva aplicação.

2.4.2 Vizinhanças variáveis

O uso sistemático de alterações da estrutura de vizinhança durante a pesquisa, constitui a base de uma meta-heurística surgida recentemente, a *Variable Neighborhood Search* (*VNS*) [Mladenovic, Hansen 1997]. A *VNS* (Algoritmo 2.8) explora o espaço de soluções através de estruturas de vizinhança cujas soluções são progressivamente mais distantes da solução actual (em termos de uma dada métrica), e recentra a pesquisa em torno de uma nova solução apenas quando a esta corresponde uma melhoria no valor da função objectivo.

As estruturas de vizinhança deverão ser ordenadas, por distâncias crescentes das soluções que lhes estão associadas em relação à solução actual. Para tal será necessária a consideração de uma *métrica do espaço de soluções*. Essa métrica será dependente da representação da solução e poderá constituir, ela própria, um elemento central no processo de concepção de estruturas de vizinhança. Com as estruturas de vizinhança disponíveis ordenadas como indicado, selecciona-se aleatoriamente uma solução na vizinhança da solução actual e aplica-se, a partir desta, um procedimento de pesquisa local. Se o óptimo local obtido for melhor do que a solução actual, esta é substituída e continua-se a pesquisa, regressando-se à primeira estrutura. Caso contrário, passa-se à estrutura seguinte.

A selecção aleatória de uma solução inicial na vizinhança da solução actual visa impedir a ocorrência de ciclos, o que poderia suceder caso fosse utilizada uma regra determinística para efectuar essa selecção.

Naturalmente que os aspectos apontados nas opções genéricas para meta-heurísticas baseadas em pesquisa local assumem aqui novamente toda a relevância. Um aprofundamento das questões específicas mais pertinentes na aplicação da *VNS* pode ser encontrado em [Hansen, Mladenovic 1999].

Algoritmo *Variable Neighborhood Search*

Calcular uma solução de partida admissível $s \in S$;

Inicializar a melhor solução encontrada $s^* = s$;

Enquanto não se cumprir o critério de paragem

Inicializar um índice de estrutura de vizinhança $v = 1$;

Enquanto $v \leq v_{max}$

Seleccionar aleatoriamente $s' \in N_v(s)$;

Realizar pesquisa local, com solução inicial s' e solução obtida s'';

Se $f(s'') < f(s)$

$s = s''$;

$v = 1$;

Se $f(s) < f(s^*)$

$s^* = s$;

Se não

$v = v + 1$;

Algoritmo 2.8 *Variable Neighborhood Search*

No presente trabalho, propõe-se a integração destes princípios noutras meta-heurísticas, designadamente o SA e a TS, com o propósito de complementar os mecanismos de intensificação e diversificação originais destas meta-heurísticas, com os mecanismos aqui descritos. Esta integração é feita de forma natural para ambos os casos: a ordem da estrutura de vizinhança é incrementada sempre que a solução obtida na exploração da vizinhança actual é de qualidade inferior à actual, e regressa-se à primeira estrutura de vizinhança no caso contrário. Quando é atingida a última estrutura de vizinhança, regressa-se à estrutura inicial. Os mecanismos originais do SA ou da TS permitem prevenir a ocorrência de ciclos. Como exemplo, apresenta-se esta integração para o caso do SA (Algoritmo 2.9). A integração com a TS seria perfeitamente idêntica.

Algoritmo *Simulated Annealing* com vizinhanças variáveis

Calcular uma solução de partida admissível $s \in S$;

Inicializar a melhor solução encontrada $s^* = s$;

Inicializar a temperatura T ;

Inicializar um índice de estrutura de vizinhança v = 1;

Enquanto não se cumprir o critério de paragem

Seleccionar aleatoriamente $s' \in N_v(s)$;

Se $f(s') < f(s)$

$s = s'$;

v = 1;

Se $f(s) < f(s^*)$

$s^* = s$;

Se não

Seleccionar aleatoriamente um número $p \in [0,1]$;

Se $p \leq e^{-\frac{f(s')-f(s)}{T}}$

$s = s'$;

Se $v = v_{max}$

v = 1;

Se não

$v = v + 1$;

Actualizar a temperatura T ;

Algoritmo 2.9 *Simulated Annealing* com vizinhanças variáveis

2.4.3 Estratégias de listas de candidatos

Designam-se por *listas de candidatos* conjuntos de movimentos "promissores", identificados, através de algum processo "inteligente", nas vizinhanças exploradas, e que são considerados para uma potencial execução em iterações subsequentes.

Entre as principais motivações para a utilização de listas de candidatos, salientam-se as seguintes [Glover, Laguna 1997, Rangaswamy et al. 1998]:

1. A eficiência e a eficácia da pesquisa podem ser positivamente influenciadas pela selecção de um conjunto de movimentos candidatos de qualidade, em contraste, por exemplo, com a avaliação de todos os movimentos numa vizinhança actual.
2. Em situações em que a geração ou a análise de cada movimento exige um elevado esforço computacional, ou quando as vizinhanças contêm um grande número de movimentos, é de todo o interesse reduzir o esforço empregue na avaliação de movimentos.
3. Apresenta igualmente particular interesse a exploração das estruturas dos problemas, em domínios particulares que permitam a construção inteligente de listas de candidatos.

As estratégias de listas de candidatos são frequentemente usadas no contexto da TS. No entanto, algumas dessas estratégias, tendo em conta as duas primeiras motivações, são gerais, não específicas a problemas ou meta-heurísticas particulares. De entre as estratégias descritas em [Glover, Laguna 1997, Rangaswamy et al. 1998], algumas, de carácter geral, têm sido destacadas em trabalhos recentes como particularmente promissoras [Fink, Voss 1999]. Neste trabalho será considerada uma dessas estratégias: a *estratégia elite* (*elite candidate list*).

O princípio em que se baseia a *estratégia elite* é o de que um conjunto dos melhores movimentos provavelmente conterá um subconjunto que continuará a manter a qualidade por várias iterações, embora não se possa prever com precisão qual será esse subconjunto. Uma monitorização adequada dos movimentos da lista de candidatos é essencial, uma vez que a alteração da solução actual pode alterar quer a sua avaliação, quer a sua admissibilidade.

A *estratégia elite* utiliza uma lista construída com os melhores m movimentos encontrados no processo de análise de movimentos alternativos numa dada iteração. Esta lista é construída periodicamente, com base na observação de um grande número de movimentos. Em cada iteração subsequente, até à criação de uma nova lista, o melhor movimento disponível na lista é escolhido e executado. O processo continua até que se tenha completado um número pré-determinado de movimentos, ou até que a qualidade do melhor movimento disponível caia abaixo de um dado limiar. Nesse ponto, a lista é reconstituída e o processo repete-se.

Em alternativa à constituição da lista com um número máximo de elementos pré-determinado, é possível a sua implementação com base num limiar mínimo, que permita garantir uma qualidade "suficiente". Com base em dados anteriores, é calculado, em cada

iteração *i*, um valor $f_{médio}$ através do amortecimento exponencial dos valores da função objectivo:

$$f_{médio}(i) = \alpha \cdot f(i) + (1 - \alpha) \cdot f_{médio}(i - 1), 0 \leq \alpha \leq 1.$$

Dependendo da avaliação do melhor movimento dentro da última vizinhança completamente avaliada f_{melhor}, é calculado um valor limiar

$$f_{limiar} = f_{médio} - \beta \cdot (f_{médio} - f_{melhor}), 0 \leq \beta \leq 1.$$

A aplicação desta estratégia no presente trabalho é feita apenas para a implementação com limite da cardinalidade da lista, e com a seguinte adaptação: é construída uma lista de candidatos permanente, que em cada iteração é actualizada com os melhores movimentos de entre os já presentes na lista e os encontrados na vizinhança actual. Desta lista de candidatos é feita a selecção do movimento a realizar, que é retirado da lista. Após a execução do movimento, são reavaliados todos os movimentos presentes na lista. Esta estratégia pode ser vista como uma vizinhança expandida, que, além da vizinhança base, considera também os elementos de uma lista, para a selecção do movimento a executar.

Também para esta estratégia se propõe a integração com TS e SA. No primeiro caso, a integração é realizada de forma directa. No segundo, a aplicação da estratégia descrita exige uma alteração ao esquema básico do SA, que consiste em considerar, em cada iteração, a pesquisa de soluções candidatas numa sub-vizinhança de dimensão superior à unidade. De outra forma, o movimento único será inserido na lista, seleccionado e retirado de seguida, não tendo a lista qualquer efeito sobre o comportamento do algoritmo.

2.4.4 Hibridização

Ao longo dos últimos anos, o interesse em meta-heurísticas híbridas tem vindo a aumentar consideravelmente. Com efeito, para muitos problemas práticos e académicos, os melhores resultados têm sido obtidos com algoritmos híbridos. Conforme já referido, o objectivo das abordagens de hibridização é tirar partido da complementaridade entre diversos princípios meta-heurísticos.

Uma taxonomia apresentada em [Talbi 1998] estabelece uma terminologia comum para esta área, bem como mecanismos de classificação, permitindo uma útil caracterização dos diversos tipos de abordagens existentes na área, e sugerindo implicitamente vários tópicos para investigação futura. A taxonomia incide sobre aspectos conceptuais e de implementação, justificando-se no contexto do presente trabalho a sua apresentação apenas na parte relativa aos aspectos conceptuais.

Nesta taxonomia, uma primeira parte, hierárquica, apresentada na Figura 2.2, compreende as seguintes distinções:

- *Baixo nível vs alto nível.*

  A hibridização de *baixo nível* tem lugar no domínio da composição funcional de um único método de optimização. Nesta classe, uma dada função de uma meta-heurística é substituída por uma meta-heurística.

  Nos algoritmos híbridos de *alto nível*, as diversas meta-heurísticas são autónomas. Não há interacção directa entre os funcionamentos internos de meta-heurísticas.

- *"Relay" vs co-evolutiva.*

  Na hibridização *"relay"*, várias meta-heurísticas são aplicadas sequencialmente, cada uma utilizando o resultado da anterior como informação de entrada.

  Uma abordagem de *co-evolução* envolve modelos de optimização cooperativos, nos quais existem múltiplos agentes cooperativos paralelos, cada um executando uma pesquisa num dado espaço de soluções.

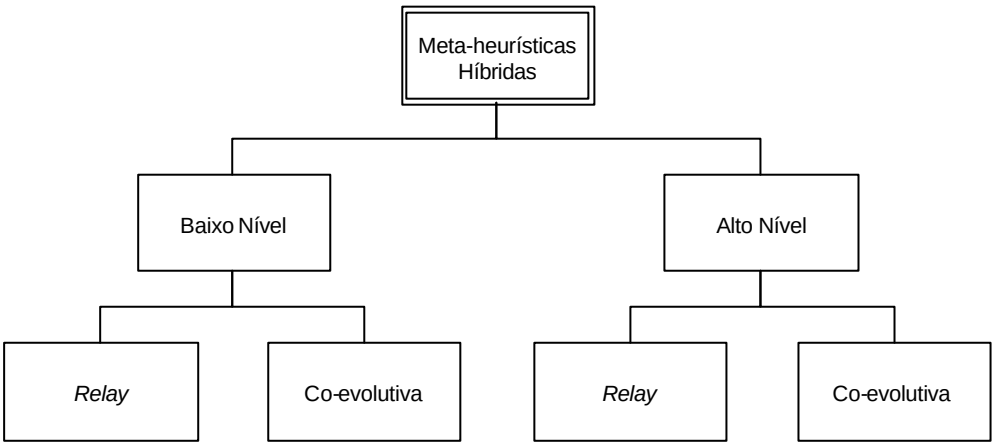


Figura 2.2 Taxonomia de abordagens híbridas - parte hierárquica

As quatro classes que resultam desta hierarquia são:

- *HBR (Híbrido Baixo Nível "Relay")*: algoritmos nos quais uma dada meta-heurística é incorporada numa meta-heurística de solução única.
- *HBC (Híbrido Baixo Nível Co-evolutivo)*: algoritmos nos quais uma dada meta-heurística é incorporada numa meta-heurística baseada em população.

- *HAR (Híbrido Alto Nível "Relay")*: meta-heurísticas autónomas executadas em sequência.
- *HAC (Híbrido Alto Nível Co-evolutivo)*: o esquema HAC envolve diversos algoritmos autónomos, realizando pesquisas em paralelo e cooperando para encontrar um óptimo. Intuitivamente, o HAC deverá comportar-se pelo menos tão bem como um algoritmo isolado, mas com maior frequência apresentará melhor comportamento, com cada um dos algoritmos a disponibilizar aos restantes informação para apoiar os respectivos processos de pesquisa.

Numa segunda parte da taxonomia, organizada horizontalmente, são consideradas as seguintes distinções:

- *Homogéneo vs heterogéneo.*

  Os algoritmos híbridos homogéneos usam repetidamente a mesma meta-heurística. Nos algoritmos heterogéneos são usadas diferentes meta-heurísticas.

- *Global vs parcial.*

  Nos algoritmos híbridos globais, todos os algoritmos componentes trabalham sobre todo o espaço de pesquisa, com o objectivo de o explorar mais exaustivamente. Nos algoritmos híbridos parciais, o problema a ser resolvido é decomposto em subproblemas, a cada um dos quais correspondendo um espaço de pesquisa próprio; cada algoritmo componente é, então, dedicado à pesquisa de um destes espaços.

- *Especialista vs Geral.*

  Nos algoritmos híbridos gerais, os algoritmos componentes resolvem o mesmo problema de optimização, enquanto os algoritmos híbridos especialistas combinam algoritmos componentes que resolvem diferentes problemas.

No contexto do presente trabalho privilegiam-se abordagens de aplicação geral e que permitam tirar partido das características de métodos já existentes, pelo que parece do maior interesse procurar estabelecer um algoritmo híbrido de alto nível, co-evolutivo, heterogéneo, global e geral.

2.4.5 Paralelização

Versões paralelas de meta-heurísticas têm vindo a ser propostas com crescente regularidade, tendo naturalmente como motivação principal a diminuição dos tempos computacionais, para instâncias de problemas com dimensões mais realistas [Crainic, Toulouse 1997]. Um segundo benefício tem sido crescentemente assinalado: em condições apropriadas, as meta-heurísticas paralelas podem ser bastante mais robustas do que as versões sequenciais, relativamente a diferenças de tipo e características de problema e às respectivas questões de afinação de parâmetros.

Sendo extenso o leque de sínteses, taxonomias e *surveys* na área, a maioria dos trabalhos, no entanto, coloca o seu enfoque numa única metodologia, ou perspectiva a área a partir de uma única metodologia. Contudo, [Crainic, Toulouse 1997] apresenta uma perspectiva mais global, propondo uma taxonomia com o objectivo de detectar aspectos comuns entre as diferentes abordagens e, assim, permitir a obtenção de *"insights"*, a descoberta de tendências e a identificação de desafios para investigação.

A classificação apresentada baseia-se no nível de impacto que a estratégia de paralelização tem na concepção algorítmica da meta-heurística:

1. *Tipo 1: Paralelização de operações dentro de uma iteração do método*. Esta estratégia, também conhecida como *paralelismo de baixo nível*, pretende apenas acelerar a computação, sem alterar o método sequencial correspondente e sem melhorar a exploração do espaço de pesquisa ou obter soluções de melhor qualidade. As avaliações de indivíduos e movimentos são operações tipicamente passíveis de paralelização do tipo 1.

2. *Tipo 2: Decomposição do domínio problema ou espaço de soluções*. Esta estratégia é baseada no princípio de que a capacidade computacional pode ser dedicada à resolução de um conjunto de problemas de dimensão mais reduzida, dos quais uma solução global melhorada pode ser extraída ou construída. A trajectória de pesquisa da abordagem paralela resultante é diferente, contudo, da do método sequencial correspondente. Este tipo de estratégias são geralmente implementadas em esquemas *"master-slave"*.

3. *Tipo 3: "Threads" multipesquisa ("multi-search") com vários graus de sincronização e cooperação*. Esta estratégia procura realizar uma exploração mais exaustiva do espaço de soluções, lançando vários *"threads"* de pesquisa que percorrem o domínio em simultâneo. Alguns aspectos variam com frequência entre implementações: utilizar uma mesma abordagem meta-heurística ou abordagens

diferentes; partir da mesma solução inicial ou de soluções iniciais distintas; realizar comunicação entre processos durante a pesquisa ou apenas no fim para identificar a melhor solução global; comunicar de forma síncrona ou assíncrona; desencadear a comunicação por eventos específicos ou em momentos pré-determinados ou decididos dinamicamente.

Estas estratégias podem ser ainda divididas em duas classes:

- *Abordagens independentes*. Diversas pesquisas são iniciadas e o melhor resultado é, no final, seleccionado de entre os resultados finais dos processos individuais.
- *Abordagens cooperativas*. São definidas de acordo com um conjunto numeroso de parâmetros: topologia de ligação, definindo a forma como os processos estão ligados; método de comunicação (*"broadcast"*, propagação, uso de memória central, etc.); processos a executar entre trocas de informação; comunicação síncrona ou assíncrona; instantes em que ocorre troca de informação; informação a trocar.

Estas abordagens têm sido aplicadas quase exclusivamente a GA, SA e TS.

Todos os tipos de estratégias de paralelização têm os seus méritos e continuarão certamente a ser aplicadas quando tal se mostrar apropriado. Parece haver, no entanto, uma clara tendência de evolução no sentido tipo 1 - tipo 2 - tipo 3.

Segundo os autores referidos, com a excepção das implementações que tiram partido de características especiais do problema ou de arquitecturas de *hardware* particulares, tem sido proposto um número limitado de estratégias de paralelização. As abordagens *"multithread"* têm-se revelado cada vez mais importantes, destacando-se a possibilidade de estes métodos permitirem não só uma mais eficiente implementação de estratégias de rearranque, mas, mais particularmente, o processamento concorrente de diferentes tipos de pesquisas - o mesmo método com diferentes parâmetros ou mesmo diferentes meta-heurísticas e métodos exactos. Desta forma, pode ser conseguida uma exploração mais exaustiva do espaço de pesquisa. Um benefício adicional é a robustez que estes métodos exibem, em comparação com os correspondentes métodos sequenciais, relativamente a diferenças nos tipos e características dos problemas.

Estas abordagens permitirão, também, vir a tirar partido do poder computacional das redes de estações de trabalho existentes em muitas organizações.

Com base nestas conclusões, e tendo em atenção o contexto particular deste trabalho, em que se pretende estudar abordagens de flexibilização gerais, sem tirar partido de uma arquitectura de *hardware* paralela e específica, fará sentido optar por uma paralelização do tipo 3, em conjunto com a opção de hibridização já referida.

2.5 Conclusões

A resolução de Problemas de Optimização Combinatória, nas mais diversas áreas, coloca exigências a que a utilização de meta-heurísticas, estratégias mestras que guiam e modificam outras heurísticas para produzir soluções de qualidade, tem respondido, com assinalável sucesso. Muito deste sucesso está relacionado com o facto de as meta-heurísticas serem ferramentas de aplicabilidade geral, com um nível de flexibilidade que permite a incorporação de restrições específicas em problemas reais, e que apresentam um conjunto interessante de compromissos entre qualidade de solução, requisitos computacionais e esforço de desenvolvimento e implementação.

Algumas das mais recentes abordagens meta-heurísticas são baseadas na flexibilização, ou seja, na introdução de mecanismos de modificação, dos seus componentes e das suas estratégias elementares. Tal sucede no âmbito da criação de novas formas de fuga à optimalidade local, da generalização de conceitos fundamentais particulares ou da utilização conjunta de conceitos provenientes de distintas meta-heurísticas.

Vários aspectos fundamentais deste trabalho são, naturalmente, reflexo do enfoque colocado neste conceito de flexibilização em meta-heurísticas: a utilização de abordagens de hibridização e paralelização de aplicação geral, a utilização de estratégias de generalização, e a identificação de uma base para infra-estruturas comuns que favoreçam a utilização dessas estratégias.

3
META-HEURÍSTICAS MULTIOBJECTIVO

Neste capítulo desenvolve-se o esforço de sistematização no domínio das meta-heurísticas, iniciado no capítulo anterior, para a área particular das meta-heurísticas multiobjectivo. Deste modo, prossegue-se o estabelecimento do contexto geral deste trabalho e completa-se a análise do domínio que irá fundamentar o desenvolvimento do *framework*.

Partindo desta sistematização, e do conjunto de tendências de flexibilização em meta-heurísticas apresentado no capítulo anterior, propõem-se, neste capítulo, no âmbito da pesquisa local multiobjectivo: um *template* em que se identificam os aspectos comuns das abordagens baseadas neste princípio, a integração das estratégias genéricas propostas no capítulo anterior, baseadas em vizinhanças variáveis e listas de candidatos, e uma abordagem de hibridização e paralelização de alto nível.

Na secção 3.1 apresentam-se os principais conceitos de Optimização Combinatória Multiobjectivo, e os métodos clássicos utilizados para a resolução de problemas de optimização desta área. A secção 3.2 consiste numa caracterização das meta-heurísticas multiobjectivo, com especial incidência nas abordagens baseadas em pesquisa local multiobjectivo. Na secção 3.3 são apresentadas as propostas referidas anteriormente: um *template* para pesquisa local multiobjectivo, a integração de estratégias genéricas, baseadas em vizinhanças variáveis e listas de candidatos, com pesquisa local multiobjectivo e uma abordagem de hibridização e paralelização de alto nível. Um pequeno apontamento conclusivo encerra o capítulo.

3.1 Optimização Combinatória Multiobjectivo

3.1.1 Introdução

A Optimização Combinatória (OC) cobre o espectro de problemas de optimização matemática com um número eventualmente muito elevado mas finito de alternativas. Há, na prática, inúmeros problemas de decisão que se podem caracterizar como sendo de OC. Por outro lado, muitos problemas reais, para serem modelados adequadamente, exigem uma avaliação das soluções segundo diversas perspectivas. Em [Steuer 1986] são apontados diversos exemplos de tais problemas, em áreas como o planeamento da produção, a gestão florestal, os transportes ou a constituição de *"portfolios"*.

O campo do conhecimento que especificamente se ocupa deste tipo de problemas é designado *"Multiple Criteria Decision Making"* (MCDM). Em [Steuer 1986] define-se MCDM como englobando todos os métodos e procedimentos através dos quais critérios múltiplos podem ser incorporados num processo analítico de tomada de decisão.

Tradicionalmente consideram-se, neste domínio, duas áreas distintas:

1. *"Multiple Attribute Decision Analysis"*, aplicável em geral a problemas com um reduzido número de alternativas e num ambiente de incerteza.
2. *"Multiple Criteria Optimization"* (*"Multiple Objective Mathematical Programming"*), tendo como objecto problemas determinísticos com um elevado (eventualmente infinito) número de alternativas.

O presente trabalho enquadra-se nesta segunda área, aqui designada por "Optimização Multiobjectivo".

Como referências fundamentais sugerem-se: [Keeney, Raiffa 1976] para a primeira das áreas indicadas e [Steuer 1986] para a segunda.

3.1.2 Optimização Multiobjectivo

O Problema de Optimização Multiobjectivo exprime-se, de forma geral, do seguinte modo:

$$\begin{aligned} \min \quad & f_1(s) \\ \min \quad & f_2(s) \\ & \vdots \\ \min \quad & f_k(s) \\ \text{com} \quad & s \in S, \end{aligned} \tag{3.1}$$

em que $f_1, f_2, ..., f_k$ são as funções objectivo, componentes de um vector função $f(s) = (f_1(s), f_2(s), ..., f_k(s))$, e S constitui a região admissível. Uma solução pode ser representada através de um vector $x = (x_1, x_2, ..., x_n), x \in \Re^n$ e $x \in S$, de *variáveis de decisão*.

Se as funções objectivo forem lineares e S for definida por restrições lineares, o problema designa-se *problema de programação linear multiobjectivo*. Com a restrição adicional de as variáveis de decisão serem inteiras, o problema será um *problema de programação linear inteira multiobjectivo*. No caso de alguma função objectivo ou restrição ser não-linear, estar-se-á perante um *problema de programação não-linear multiobjectivo*. Se se impuser a restrição de integralidade das variáveis de decisão, será um *problema de programação não-linear inteira multiobjectivo*.

Nos problemas de optimização multiobjectivo consideram-se habitualmente, para efeitos de representação, dois espaços distintos: o *espaço dos objectivos* e o *espaço das variáveis de decisão*. A formulação (3.1) é realizada no espaço das variáveis de decisão. Enquanto S representa a região admissível no espaço das variáveis de decisão,

$$Z = \left\{ z = (z_1, z_2, ..., z_k) \in \Re^k : z_i = f_i(s), s \in S \right\} \tag{3.2}$$

representa a região admissível no espaço dos objectivos. A z chama-se *vector de critérios*. Desta forma, o problema de optimização multiobjectivo pode ser formulado no espaço dos objectivos da seguinte forma:

$$\begin{aligned} \min \quad & z_1 \\ \min \quad & z_2 \\ & \vdots \\ \min \quad & z_k \\ \text{com} \quad & z \in Z. \end{aligned} \tag{3.3}$$

Facilmente se verifica que, excluindo o caso (naturalmente não interessante, na prática) em que existe uma solução admissível que simultaneamente optimiza todos os k objectivos, não tem sentido, para este problema, o conceito de solução óptima. Segundo [Steuer 1986], a forma ideal de "resolver" um problema multiobjectivo seria determinar a *função de utilidade* $U : \Re^k \rightarrow \Re$ do decisor, que realiza o mapeamento dos vectores de critérios em $\Re$ de forma a que a maiores valores de $\Re$ corresponda uma mais forte preferência do decisor, e resolver o problema

$$\begin{aligned} \max \quad & \{U(z_1, z_2, ..., z_k)\} \\ \text{com} \quad & f_i(s) = z_i, 1 \leq i \leq k \\ & s \in S. \end{aligned} \tag{3.4}$$

Este problema seria provavelmente não-linear. Contudo, esta não será a principal dificuldade da abordagem, que consiste, antes, na impossibilidade, para muitos problemas, de obter uma representação matemática da função de utilidade do agente de decisão (ou, por outras palavras de estabelecer uma "relação de troca" entre os objectivos que represente adequadamente as preferências do agente de decisão).

Assim, a abordagem mais natural para "resolver" o problema consiste em pesquisar o "espaço de compromissos" entre os objectivos, à procura da solução "óptima" do agente de decisão, usando apenas informação implícita. Por outro lado, a utilização da função de utilidade enquadra-se numa escola de pensamento no campo da MCDM - a *escola americana* - fortemente criticada pela *escola francesa*, segundo a qual é necessário evoluir dos métodos que procuram a solução "óptima", para os métodos orientados para a aprendizagem, procurando esclarecer o decisor acerca do problema e das suas próprias preferências [Vincke 1989, Vincke 1995]. Não existem, portanto, neste domínio, e até ao momento, abordagens inquestionáveis e definitivas.

3.1.3 Definições e conceitos fundamentais de Optimização Multiobjectivo

Os conceitos e definições fundamentais de Optimização Multiobjectivo, relevantes para o presente trabalho, serão (para problemas de minimização) sucintamente apresentados nesta secção, com base em [Steuer 1986].

Dominância e eficiência

Definição 3.1: Sejam $z \in Z$ e $z' \in Z$. Diz-se que z *domina* z' se e só se $z \leq z'$ e $z \neq z'$, isto é, $z_i \leq z'_i, \forall i$ e $z_i < z'_i$, para algum i. Caso contrário, diz-se que z' é *não-dominada* por z.

Definição 3.2: Uma solução $z \in Z$ diz-se *não-dominada* se e só se não existir outra solução $z' \in Z$ que a domine. Caso contrário z diz-se *dominada*.

Definição 3.3: Uma solução $s \in S$ diz-se *eficiente* se e só se a sua imagem no espaço dos objectivos for uma solução não dominada. Designações alternativas são *óptima de Pareto* e *não-inferior*. Caso contrário s diz-se *não-eficiente.*

Ideal e anti-ideal

Definição 3.4: O *ponto ideal* $z^* = (z^*_1, z^*_2, ..., z^*_k)$ obtém-se através da optimização individual de cada função objectivo, ou seja,

$$z^*_i = \min_{z \in Z} z_i, \forall i .$$

Definição 3.5: A *tabela de "payoff"* é uma tabela em que cada coluna corresponde a uma função objectivo e cada linha ao vector de critérios de cada solução não-dominada que optimiza cada critério individualmente. Genericamente o elemento z_{ji} da tabela representa o valor da i-ésima função objectivo quando se optimiza individualmente a j-ésima função objectivo.

Definição 3.6: O *ponto nadir* ou *ponto anti-ideal* $n^* = (n^*_1, n^*_2, ..., n^*_k)$ é o vector constituído pelos maiores valores que cada função objectivo assume na tabela de *payoff*, isto é,

$$n^*_i = \max_{\substack{j=1,...,k \\ j \neq i}} z_{ji}, \forall i .$$

Solução suportada e não-suportada

Sejam X e Y conjuntos em $\Re^n$. Então, a *adição de conjuntos* de X e Y (denotada $X \oplus Y$) é dada por

$$X \oplus Y = \{z \in \Re^n \mid z = x + y, x \in X, y \in Y\},$$

ou seja, todo o ponto em X é adicionado a todo o ponto em Y.

Seja N o conjunto das soluções não-dominadas e $Z^{\geq}$ o invólucro convexo (*"convex hull"*) de $[N \oplus \{z \in \Re^k \mid z \geq 0\}]$.

Definição 3.7: Uma solução $z \in N$ diz-se *suportada* se estiver sobre a fronteira de $Z^{\geq}$. Caso contrário z diz-se *não-suportada* ou *dominada de forma convexa.*

Definição 3.8: Uma solução suportada $z \in N$ diz-se *extrema suportada* se for um ponto extremo de $Z^{\geq}$. Caso contrário z diz-se *não-extrema suportada*.

As soluções no espaço das variáveis de decisão a que correspondem soluções não-dominadas suportadas são chamadas *soluções eficientes suportadas*. As soluções no espaço das variáveis de decisão a que correspondem soluções não-dominadas não-suportadas são chamadas *soluções eficientes não-suportadas.*

Métricas

Definição 3.9: Uma métrica em $\Re^n$ é uma função de distância que atribui a cada par de vectores $x, y \in \Re^n$ um escalar $\|x - y\| \in \Re$ se e só se a função satisfaz, para todo o $z \in \Re^n$ os seguintes quatro axiomas:

1. $\|x - y\| \geq 0$ e $\|x - x\| = 0$.
2. $\|x - y\| = \|y - x\|$.
3. $\|x - y\| \leq \|x - z\| + \|z - y\|$.
4. Se $x \neq y$, então $\|x - y\| > 0$.

Definição 3.10: Para a *família de métricas* L_p, as funções de distância são dadas pela seguinte expressão:

$$\|x - y\|_p = \left[\sum_{i=1}^{n} |x_i - y_i|^p \right]^{1/p} \qquad p \in \{1, 2, 3, \ldots\} \cup \infty .$$

Definição 3.11: A métrica L_∞ ou *métrica de Tchebycheff* é dada por

$$\|x - y\|_\infty = \max_i [|x_i - y_i|]. \tag{3.5}$$

Definição 3.12: Para a *família de métricas* L_p *ponderadas* as funções de distância são dadas pela seguinte expressão:

$$\|x - y\|_p^{\lambda} = \left[\sum_{i=1}^{n} \left(\lambda_i |x_i - y_i|^p \right) \right]^{1/p} \quad p \in \{1, 2, 3, \ldots\} \cup \infty ,$$

sendo $\lambda \in \Re^n$ um vector de pesos não-negativos.

Definição 3.13: A métrica $L_{¥}$ *ponderada* ou *métrica de Tchebycheff ponderada* é dada por

$$\|x-y\|_{\infty}^{ë} = \max_i \left[l_i |x_i - y_i| \right]. \tag{3.6}$$

Funções escalarizantes

Sejam

$$\Lambda = \left\{ ë \in \Re^k \mid \lambda_i > 0, \sum_{i=1}^{k} \lambda_i = 1 \right\}$$

o conjunto de todos os *vectores de pesos estritamente positivos* e

$$\overline{\Lambda} = \left\{ ë \in \Re^k \mid \lambda_i \geq 0, \sum_{i=1}^{k} \lambda_i = 1 \right\}$$

o conjunto de todos os *vectores de pesos não-negativos*.

Definição 3.14: Uma *função escalarizante* é uma função que realiza um mapeamento de Z em $\Re$.

Definição 3.15: A função escalarizante *soma ponderada* é dada pela seguinte expressão:

$$e_{sp}(z, ë) = \sum_{i=1}^{k} l_i z_i, ë \in \Lambda \cdot \tag{3.7}$$

Seja um vector $z^{**} = (z^{**}_1, z^{**}_2, ..., z^{**}_k)$, obtido a partir dos componentes do ponto ideal z^* da seguinte forma: $z^{**}_i = z^*_i - e_i, \forall i$, com $e_i \geq 0$.

Definição 3.16: A *função escalarizante de Tchebycheff ponderada* é dada pela seguinte expressão:

$$e_{Tp}(z, ë) = \max_i \left\{ l_i \left(z_i - z^{**}_i \right) \right\}, ë \in \overline{\Lambda}. \tag{3.8}$$

Seja um parâmetro r com um valor positivo suficientemente pequeno, mas numericamente significativo.

Definição 3.17: A *função escalarizante de Tchebycheff ponderada aumentada* é dada pela seguinte expressão:

$$e_{Tpa}(z, ë) = e_{Tp}(z, ë) + r \sum_{i=1}^{k} \left(z_i - z^{**}_i \right), ë \in \overline{\Lambda} \cdot \tag{3.9}$$

Problema de soma ponderada

Definição 3.18: Designa-se *problema de soma ponderada* o seguinte problema:

$$\begin{aligned} \min \quad & e_{sp}(z, \ddot{e}) \\ \text{com} \quad & z \in Z . \end{aligned} \tag{3.10}$$

Este problema goza das seguintes propriedades:

1. Para cada vector de pesos $\mathbf{l}$, a solução óptima de (3.10) é não-dominada.
2. Para cada solução não-dominada suportada z, existe um vector de pesos, $\mathbf{l}$, tal que z é a solução óptima de (3.10).

Em problemas de optimização linear multiobjectivo não existem soluções não-dominadas não-suportadas. Contudo, em problemas de optimização inteira ou não-linear multiobjectivo, ocorrem frequentemente soluções não-dominadas que são não-suportadas. Uma vez que cada membro não-suportado de N é dominado por alguma combinação convexa de outras soluções não-dominadas, *não é possível, neste caso, gerar soluções não-suportadas usando o problema de soma ponderada,* independentemente do $\mathbf{l} \in \Lambda$ utilizado.

Problema de Tchebycheff ponderado aumentado

Definição 3.19: Designa-se *problema de Tchebycheff ponderado aumentado* o seguinte problema:

$$\begin{aligned} \min \quad & e_{Tpa}(z, \ddot{e}) \\ \text{com} \quad & z \in Z . \end{aligned} \tag{3.11}$$

Este problema pode igualmente ser escrito de forma linear:

$$\begin{aligned} \min \quad & a + r \sum_{i=1}^{k} \left(z_i - z^{**}_i \right) \\ \text{com} \quad & a \geq l_i \left(z_i - z^{**}_i \right), \forall i \\ & z \in Z . \end{aligned} \tag{3.12}$$

Uma abordagem baseada neste problema apresenta vantagens sobre o problema de soma ponderada, na medida em que permite a determinação de soluções não-suportadas não-dominadas. O problema goza das seguintes propriedades:

1. Para cada vector de pesos $\mathbf{l}$, a solução óptima de (3.11) é não-dominada.

2. Para cada solução não-dominada z, existe um vector de pesos, $\boldsymbol{l}$, e um factor de aumento, r, tal que z é a solução óptima de (3.11).

Em geral será suficiente que, para a definição de z^{**}, cada e_i seja positivo e tenha um valor moderadamente reduzido. No entanto, em muitos, mesmo na maioria, dos problemas multiobjectivo, é aceitável que um ou mais dos e_i sejam zero. A obrigatoriedade de e_i ser positivo sucede aquando da sua utilização com instâncias de problemas em que:

1. existe mais do que um vector de critérios a optimizar o objectivo i ou
2. existe apenas um vector de critérios que optimiza o objectivo i, mas esse vector de critérios optimiza também algum dos restantes objectivos.

3.1.4 Optimização Combinatória Multiobjectivo

O Problema de Optimização Combinatória Multiobjectivo é um problema de optimização combinatória em que são considerados vários objectivos, e exprime-se, de forma geral, do seguinte modo:

$$\begin{aligned} \min \quad & f_1(s) \\ \min \quad & f_2(s) \\ & \vdots \\ \min \quad & f_k(s) \\ \text{com} \quad & s \in S, \ S \text{ discreto,} \end{aligned} \tag{3.13}$$

em que $f_1, f_2, ..., f_k$ são as funções objectivo, componentes de um vector função $f(s) = (f_1(s), f_2(s), ..., f_k(s))$, e S constitui a região admissível.

Em [Czyzak, Jaszkiewicz 1998] e [Hansen 1998] são referidas inúmeras aplicações práticas dos métodos surgidos nesta área, em domínios como o sequenciamento de actividades, transportes ou *design* de redes. Uma recente bibliografia anotada, dedicada a esta área, pode ser encontrada em [Ehrgott, Gandibleux 2000].

A dificuldade deste tipo de problemas advém de dois factores principais [Czyzak, Jaszkiewicz 1998]:

1. O processo de resolução deverá sempre envolver interacção com o agente de decisão, facto que coloca fortes exigências à eficiência das ferramentas utilizadas para gerar soluções eficientes.

2. Muitos problemas combinatórios são já difíceis nas suas versões com um só objectivo; as versões multiobjectivo são frequentemente muito mais difíceis.

Para um aprofundamento das questões de complexidade computacional em Optimização Combinatória Multiobjectivo, refere-se [Serafini 1987] e, mais recentemente, [Ehrgott 2000]. Destacar-se-á apenas que o estudo da complexidade computacional nesta área se baseia numa definição [Serafini 1987] do problema de decisão correspondente ao problema de optimização combinatória multiobjectivo: para um determinado vector de critérios z, existe uma solução $s \in S$ tal que $f(s) \leq z$?

Tal como na OC com um só objectivo, também na área multiobjectivo as abordagens abrangem desde os métodos exactos às heurísticas especializadas e meta-heurísticas, passando por todo um conjunto de outras abordagens que visam a redução do esforço computacional empregue na resolução dos problemas.

3.1.5 Métodos clássicos para Optimização Combinatória Multiobjectivo

Os métodos clássicos para Optimização Combinatória Multiobjectivo baseiam-se na utilização de métodos exactos para um só objectivo, enquadrados em abordagens baseadas em funções de agregação ou prioritização de objectivos.

Uma apresentação sucinta de alguns dos principais métodos clássicos é realizada nesta secção, com base em [Steuer 1986], [Viana 1997] e [Coello 1999].

Abordagens baseadas em funções de agregação

Trata-se de abordagens que, na base, são simples, mas que, no entanto, quando aplicadas no domínio da OC, apresentam elevadas cargas computacionais. Um problema geral que apresentam é a necessidade de se conhecer as gamas de valores dos objectivos, por forma a evitar que uns dominem os outros.

- *Método do Equivalente Paramétrico.* Este método baseia-se na resolução do problema de soma ponderada (3.10), tendo sido a primeira técnica desenvolvida para geração de soluções não-dominadas em optimização multiobjectivo.

  Verificando-se os pressupostos de convexidade do espaço de soluções e concavidade das funções objectivo, uma solução óptima do problema de soma ponderada será uma solução eficiente do problema multiobjectivo.

Para obter o conjunto das soluções eficientes, procede-se a uma variação sistemática dos pesos, o que no entanto, conforme justificado a propósito de (3.10), não permite gerar soluções não-dominadas não-suportadas, podendo tal suceder nos problemas de programação inteira ou não-linear multiobjectivo.

- *"Goal Programming"*. Neste método, o agente de decisão atribui a cada um dos objectivos *i* uma meta M_i que deseja atingir. Estes valores são incorporados no problema como restrições adicionais. É definida uma nova função objectivo com a qual se procura então minimizar os desvios absolutos entre os objectivos e as metas.

  A formulação mais geral da função objectivo desta abordagem é:

$$\min \quad \sum_{i=1}^{k} \left| f_i(s) - M_i \right| \qquad (3.14)$$

$$\text{com} \quad s \in S.$$

  Um inconveniente deste método reside no facto de que a solução gerada será dominada se a meta for um ponto da região admissível no espaço dos objectivos. A técnica será sobretudo útil nos casos em que é possível realizar uma aproximação linear das funções objectivo.

- *"Goal Attainment"*. Nesta abordagem, o agente de decisão deve, para além das metas desejadas M_i, especificar um vector de pesos w_i normalizados, que estabelece uma relação entre o maior ou menor grau com que se pretende sub- ou sobre-atingir cada uma das metas desejadas. Se algum $w_i = 0$, o limite máximo para $f_i(s)$ é M_i.

  Para encontrar a melhor solução de compromisso, resolve-se o seguinte problema:

$$\begin{aligned} &\min && a \qquad (3.15)\\ &\text{com} && M_i + a \cdot w_i \geq f_i(s),\ 1 \leq i \leq k \\ & && s \in S. \end{aligned}$$

  O conjunto das soluções não-dominadas pode ser gerado por variação dos pesos $w_i \geq 0$, mesmo para problemas não-convexos.

O valor óptimo de α permite ao agente de decisão reconhecer se os objectivos são atingíveis ou não. Um valor negativo implica que os objectivos são atingíveis; um valor positivo, o contrário.

Um ponto fraco desta abordagem é o facto de poder induzir o agente de decisão em erro. Por exemplo, é possível que num problema biobjectivo duas soluções candidatas, tendo o mesmo valor num objectivo e valores diferentes no outro, tenham igual valor de α, o que significa que nenhuma é considerada melhor que a outra.

- *"e-constraint"*. Este método é baseado na minimização de uma função objectivo (*primária*), considerando os outros objectivos como restrições limitadas por níveis permitidos e_i.

$$\begin{aligned} \min \quad & f_r(s) \\ \text{com} \quad & f_i(s) \le e_i,\ 1 \le i \le k, i \ne r \\ & s \in S. \end{aligned} \tag{3.16}$$

Consequentemente, é realizada uma minimização de um só objectivo para a função seleccionada, sujeita a restrições sobre todas as outras funções objectivo. Os níveis e_i são então alterados para gerar todo o conjunto de soluções eficientes. Quando os níveis são muito baixos, o problema (3.16) não possui soluções admissíveis, e pelo menos um dos níveis necessita de ser relaxado.

De acordo com o contexto em que se realiza a tomada de decisão, este processo pode ser efectuado tomando para função objectivo primária apenas uma ou, sucessivamente, algumas ou todas as funções objectivo.

O método é bastante geral e simples, apresentando-se no entanto como computacionalmente exigente e com problemas de operacionalização, devido à dificuldade em sistematizar a evolução dos níveis permitidos.

Abordagens baseadas em prioritização

A forma mais conhecida deste tipo de abordagens é o *Método Lexicográfico*, em que o agente de decisão ordena os objectivos pela importância que lhes atribui. A solução "óptima" é então obtida por minimização das funções objectivo, começando pela mais importante e continuando de acordo com a ordem de importância atribuída aos objectivos, garantindo que os valores das funções objectivo já minimizadas não sejam excedidos.

A simplicidade destas abordagens tem naturalmente como desvantagem o facto de a relação de compromisso entre os objectivos ser muito elementar, e dificilmente representativa das preferências do agente de decisão.

Desvantagens comuns aos vários métodos

Em [Viana 1997] são apontadas as principais desvantagens destes métodos:

1. Em geral, os procedimentos para geração de todo o conjunto de soluções eficientes são muito lentos.
2. A obtenção de uma função de avaliação explícita pode ser difícil, quer em termos de tempo quer de esforço despendido.
3. O facto de se optimizar apenas um objectivo restringe o conhecimento do tipo de soluções que se podem obter para o problema.
4. A sensibilidade aos pesos ou critérios de aspiração torna necessário um estudo cuidado do problema antes da construção da função de agregação.
5. Alguns métodos desta classe têm pressupostos rigorosos para a sua aplicação, como, por exemplo, a exigência de convexidade do espaço de soluções, no caso do método do equivalente paramétrico.

3.2 Meta-heurísticas multiobjectivo

Referiram-se já, no capítulo anterior, as características das meta-heurísticas que as tornam ferramentas interessantes na resolução de problemas de OC. Essas mesmas características potenciam também a sua aplicação na Optimização Combinatória Multiobjectivo, com especial relevância para a flexibilidade que, em particular, permite tratar mudanças ao nível da formulação do problema ou da família de objectivos.

Conforme refere [Hansen 1998], a finalidade da MCDM não consiste em obter uma aproximação do conjunto de soluções não-dominadas, e muito menos esse conjunto. Consiste, antes, em obter *a* solução preferida, que obviamente deverá ser seleccionada de entre as soluções não-dominadas (ou de um conjunto que seja aproximação a essas soluções). As meta-heurísticas multiobjectivo diferenciam-se das meta-heurísticas para um só objectivo essencialmente pelo facto de serem especificamente desenhadas para gerar soluções (exacta ou aproximadamente eficientes), de forma integrada, num procedimento interactivo.

Como é natural, tratando-se de uma linha de investigação recente, as abordagens nesta área têm-se desenvolvido essencialmente em torno das meta-heurísticas "clássicas": GA, SA e TS. Uma referência a outras abordagens pode, no entanto, ser encontrada em [Hansen 1998].

3.2.1 Algoritmos Genéticos

Um levantamento exaustivo e pormenorizado do trabalho desenvolvido nesta área pode ser obtido em [Fonseca, Fleming 1995] e [Coello 1999]. Nos trabalhos referidos, os seguintes factores são considerados determinantes para o pioneirismo dos GA como meta-heurística multiobjectivo:

1. o facto de se trabalhar, em cada momento, com uma população de soluções, o que permite procurar o conjunto de soluções eficientes numa única execução do algoritmo, em vez de se realizar um conjunto de execuções separadas, como é o caso das abordagens de programação matemática tradicionais;
2. a menor susceptibilidade à forma ou continuidade da fronteira de Pareto, que constituem preocupações fundamentais nas técnicas de programação matemática.

Com base nas referências citadas, apresenta-se de seguida uma descrição necessariamente sucinta de algumas das abordagens mais relevantes nesta área, estruturadas em três grandes grupos:

1. baseadas em funções de agregação;
2. outras, não baseadas na noção de óptimo de Pareto;
3. baseadas na noção de óptimo de Pareto.

Abordagens de GA baseadas em funções de agregação

As abordagens *baseadas em funções de agregação* surgem naturalmente ligadas ao facto de os GA utilizarem informação de aptidão (*"fitness"*) escalar, inspirando-se nos diversos métodos clássicos que empregam funções de agregação (e mantendo as suas vantagens e inconvenientes):

- *Equivalente Paramétrico* [Syswerda, Palmucci 1991],
- *"Goal Programming"* [Wienke et al. 1992],
- *"Goal Attainment"* [Wilson, MacLeod 1993] e
- *"e-constraint"* [Ritzel et al. 1994].

Um exemplo particular de utilização de funções de agregação, neste caso aditivas, e com grande interesse na área das meta-heurísticas, é a incorporação de restrições na função objectivo, considerando penalizações pela sua violação.

Outras abordagens de GA, não baseadas na noção de óptimo de Pareto

Deve, antes do mais, referir-se que uma preocupação comum a diversas abordagens aqui enquadradas consiste em procurar evitar a convergência para uma única região da fronteira de Pareto. Para tal, o mecanismo utilizado mais frequentemente é o *"fitness sharing"* [Goldberg, Richardson 1987], um mecanismo de criação de "nichos" (*"niches"*), que permite manter soluções ao longo de toda a fronteira de Pareto. Este mecanismo realiza ajustes ao valor das aptidões das soluções, penalizando as que se encontram em áreas de grande "densidade" de soluções. Em torno de cada solução a avaliar é definido um "nicho" de determinada dimensão, sendo a "densidade" aferida não só pelo número de soluções aí encontradas, mas também pela sua proximidade à solução a avaliar.

O conjunto de outras abordagens *não baseadas na noção de óptimo de Pareto*, a que se refere esta secção, fundamenta-se em dois princípios: o tratamento separado dos objectivos e a consideração de objectivos alternativos. Apresentam-se de seguida algumas destas abordagens.

1. Tratamento separado dos objectivos

- Em [Schaffer 1985] foi introduzido o *VEGA* (*Vector Evaluated Genetic Algorithm*), algoritmo em que se realizam diversas selecções de soluções, segundo cada um dos objectivos, dando origem a diversas sub-populações que são posteriormente fundidas numa única população.
- O *Método Lexicográfico* encontrou aplicação na selecção de soluções realizada por comparação de pares, de acordo com as prioridades atribuídas aos objectivos pelo agente de decisão [Fourman 1985]. Uma implementação alternativa recorre à selecção aleatória, em cada iteração, do objectivo a utilizar para comparação [Kursawe 1991].
- Em [Périaux et al. 1997] é apresentada uma abordagem baseada em *Teoria dos Jogos*, aplicada a um problema biobjectivo. Duas sub-populações independentes são vistas como jogadores não-cooperativos que, em cada geração, tentam optimizar um dos objectivos, sem degradar o outro, sendo a melhor solução de cada uma das sub-populações enviada para a outra.

- Diversas abordagens têm utilizado o conceito de *"géneros"*, identificados com objectivos. No caso mais geral [Lis, Eiben 1996], a recombinação envolve várias soluções geradoras, uma de cada género. O género da solução gerada é idêntico ao da solução geradora mais determinante para as suas características. Para cada género há diferentes funções de aptidão e o operador de mutação é concebido de forma a não alterar o género das soluções.

Nos casos referidos, com excepção da abordagem baseada em *Teoria dos Jogos*, na qual se gera apenas uma solução não-dominada, procede-se, durante a execução do algoritmo, à identificação de soluções não-dominadas. Esta informação não é, no entanto, utilizada pelos algoritmos.

2. Utilização de objectivos alternativos

- Em [Hajela, Lin 1992] as soluções são avaliadas por comparação com o ponto ideal, determinado pela resolução separada dos problemas de optimização para cada critério. Nessa avaliação, é utilizada uma abordagem *min-max*, sendo considerada como solução desejável aquela que apresentar os menores incrementos relativos, face aos extremos de cada função objectivo. Na referência citada são utilizadas diversas combinações de pesos dentro da população. Noutras referências, diversas variações têm sido propostas, como a utilização de pesos aleatórios, a evolução paralela de várias populações com conjuntos de pesos distintos, ou a aplicação da abordagem *min-max* na selecção por comparação de pares.

- Uma abordagem baseada na determinação de distâncias relativas ao conjunto de Pareto é apresentada em [Osyczka, Kundu 1995]. As soluções não dominadas têm uma avaliação unitária, enquanto a solução mais distante de todas as soluções não-dominadas, de acordo com uma métrica euclidiana (L_2), tem uma avaliação nula. Para as restantes soluções, as avaliações são atribuídas por interpolação linear entre zero e um, com base na respectiva distância euclidiana à solução não-dominada mais próxima.

- Em [Valenzuela-Rendón, Uresti-Charre 1997] o problema original é transformado num problema com dois objectivos: minimizar um indicador de dominância, constituído pela média ponderada do número de soluções que dominaram o indivíduo até ao momento; minimizar um contador de "nicho", dado pela média ponderada do número de indivíduos considerados próximos segundo uma certa função de *"sharing"*. Este problema é transformado num problema de um só objectivo por combinação linear dos dois objectivos.

Abordagens de GA baseadas na noção de óptimo de Pareto

Em [Goldberg 1989] foi proposta a utilização de uma função de aptidão baseada na optimalidade de Pareto. Esta abordagem opera com base no operador de selecção e numa *classificação de não-dominância*: determinam-se as soluções não-dominadas, que recebem a classificação mais elevada e são removidas da população; a operação repete-se até toda a população estar classificada. Para impedir a convergência para uma única solução (*"genetic drift"*), recorre-se, conforme já referido, a um mecanismo de criação de "nichos".

Uma das maiores dificuldades com que se depara este tipo de abordagem é a inexistência de algoritmos eficientes para verificar a não-dominância num conjunto de soluções admissíveis. A determinação de uma dimensão de nicho adequada apresenta também algumas dificuldades.

Neste conjunto de abordagens é ainda possível destacar:

- *"Multiple Objective Genetic Algorithm"* [Fonseca, Fleming 1993]. A classificação de uma solução corresponde, neste caso, ao número de soluções pelos quais a solução é dominada. As soluções são ordenadas de acordo com a sua classificação, sendo-lhes atribuída uma aptidão, através de interpolação, linear ou não, da melhor à pior classificação. Para todas as soluções com a mesma classificação é calculada uma aptidão média, de forma a que possam ser objecto de amostragem com a mesma probabilidade.

  Para evitar a convergência prematura, recorre-se a métodos de formação de nichos, através de *"fitness sharing"* no espaço dos objectivos. Este facto impede a existência na população de duas soluções diferentes com vectores de critérios idênticos.

- *"Non-dominated Sorting Genetic Algorithm"* [Srinivas, Deb 1993]. Este algoritmo procede de forma semelhante ao anterior, mas recorrendo à classificação proposta em [Goldberg 1989].

- *"Niched Pareto Genetic Algorithm"* [Horn, Nafpliotis 1993]. A selecção é, neste caso, baseada na comparação de pares de soluções, mas tomando em consideração um conjunto de outras soluções. Se ambas as soluções forem dominadas ou não-dominadas, o resultado decide-se por *"fitness sharing"*.

3.2.2 *Simulated Annealing*

Destacam-se na literatura três meta-heurísticas multiobjectivo baseadas em SA. As duas primeiras foram propostas em [Serafini 1992] e [Fortemps et al. 1994], de forma independente, mas com fortes semelhanças. Ambas se baseiam em execuções repetidas de SA com um só objectivo, de acordo com o esquema geral apresentado no Algoritmo 3.1.

Algoritmo *Simulated Annealing* Multiobjectivo

Calcular uma solução de partida admissível $s \in S$;

Inicializar a aproximação ao conjunto eficiente $E = \{\}$;

Actualizar E com s;

Inicializar a temperatura T;

Enquanto não se cumprir o critério de paragem

Seleccionar aleatoriamente $s' \in N(s)$;

Se $f(s')$ não-dominada por $f(s)$

Actualizar E com s';

Seleccionar aleatoriamente um número p no intervalo [0; 1];

Se $p \leq P(s, s', T, \lambda)$

$s = s'$;

Actualizar a temperatura T;

Algoritmo 3.1 *Simulated Annealing* Multiobjectivo

A questão fulcral nos algoritmos com esta estrutura reside na definição da função de probabilidade de aceitação $P(s, s', T, \lambda)$. Para tal torna-se necessário abordar dois tipos de questões:

1. *A distinção entre movimentos melhoradores e movimentos não-melhoradores.*

   É possível a ocorrência de três tipos de movimentos:

   - apresentando melhorias em todos os objectivos;
   - apresentando melhorias em alguns objectivos e degradação noutros;
   - apresentando degradação em todos os objectivos.

As distinções possíveis seriam, então:

- *Regra fraca*: considerar que o primeiro e o segundo tipos de movimentos são movimentos melhoradores.
- *Regra forte*: considerar que o segundo e o terceiro tipos de movimentos são não-melhoradores.

A primeira regra, segundo [Ulungu et al. 1997], apresenta vantagens do ponto de vista da diversidade, mas tipicamente não permite atingir bons compromissos entre os objectivos, conduzindo a soluções longe da fronteira eficiente. Estes autores têm, consequentemente, privilegiado a utilização da segunda regra.

2. *A distância entre soluções*.

Haverá aqui duas abordagens possíveis:

- *Escalarização dos objectivos*, por exemplo com as métricas de soma ponderada (L_1 ponderada) ou de Tchebycheff (L_∞) ponderada (3.6).
- *Escalarização das probabilidades*. Seja

$$p_k = \min\left\{1, e^{-\frac{\Delta z_k}{T}}\right\}$$

a probabilidade de aceitação para cada um dos objectivos k. Constituem possibilidades naturais para a referida escalarização:

- o mínimo das probabilidades calculadas, com ponderação, separadamente em cada um dos objectivos, $\min_k (p_i)^{l_i}$,
- ou o produto das mesmas, $\prod_{i=1}^{k} (p_i)^{l_i}$.

Em [Ulungu et al. 1997] demonstra-se que há equivalência entre estas abordagens:

- A escalarização de objectivos através da soma ponderada tem o mesmo comportamento que a escalarização através do produto das probabilidades ponderadas individuais.
- A escalarização através da métrica de Tchebycheff ponderada tem o mesmo comportamento que a escalarização através do mínimo das probabilidades ponderadas individuais.

A escalarização envolve a utilização de pesos, que pode realizar-se num contexto em que o agente de decisão manifesta as suas preferências, ou então numa repetição da execução do algoritmo com conjuntos bastante diferentes e dispersos dos pesos, por forma a gerar uma aproximação da fronteira não-dominada.

A terceira abordagem, mais recente, designada *Pareto Simulated Annealing* (PSA) [Czyzak, Jaskiewicz 1998], baseia-se na utilização de uma população de soluções geradoras e no controlo dos pesos dos objectivos, por forma a assegurar a dispersão das soluções geradoras ao longo de toda a fronteira de Pareto. O mecanismo que assegura esta dispersão é a probabilidade de aceitação, com base no facto de que um aumento dos pesos associados a um objectivo reduz a probabilidade de aceitação de movimentos que não melhoram esse objectivo e aumenta a probabilidade de esse objectivo melhorar. Os pesos são actualizados com um factor multiplicativo α ou o seu inverso $1/\alpha$, sendo α uma constante com um valor superior à unidade, mas próximo desta (por exemplo, 1.05). Na determinação da solução mais próxima é possível utilizar várias métricas, como a soma ponderada ou a métrica de Tchebycheff ponderada.

Um aspecto crítico deste tipo de abordagens é a actualização da aproximação ao conjunto eficiente: uma solução será inserida no conjunto se não for dominada por nenhuma das suas soluções; as soluções do conjunto que passem, em virtude desta inserção, a ser dominadas, devem ser removidas. Esta actualização, que é realizada com uma frequência muito elevada no decorrer da execução do algoritmo, pode ser computacionalmente pesada, em particular em casos em que o número de elementos da aproximação ao conjunto eficiente seja elevado. Em [Czyzak, Jaskiewicz 1998] é sugerida a utilização de *"quad trees"* ([Finkel, Bentley 1974] e [Habenicht 1982]) para uma implementação eficiente deste procedimento.

Algoritmo *Pareto Simulated Annealing*

Calcular um conjunto de soluções de partida admissíveis $G \subset S$;

Inicializar a aproximação ao conjunto eficiente $E = \{\}$;

Para cada $s \in G$

Actualizar E com s;

Inicializar a temperatura T;

Enquanto não se cumprir o critério de paragem

Para cada $s \in G$

Seleccionar a solução $t \in G$ mais próxima de s e não-dominada

por esta;

Se não existir tal solução **ou** é a primeira iteração com s

Atribuir aos elementos do vector de pesos λ associado a s

valores aleatórios tais que $\lambda_i \geq 0, \forall i \text{ e } \sum_{i=1}^{k} \lambda_i = 1$;

Se não

Para cada objectivo f_i

$$\lambda_i = \begin{cases} a\, l_i, & \text{se } f_i(s) \leq f_i(t) \\ l_i / a, & \text{se } f_i(s) > f_i(t) \end{cases};$$

Normalizar os pesos de forma a que $\sum_{i=1}^{k} \lambda_i = 1$;

Seleccionar aleatoriamente $s' \in N(s)$;

Se $f(s')$ não-dominada por $f(s)$

Actualizar E com s';

Seleccionar aleatoriamente um número p no intervalo [0; 1];

Se $p \leq P(s, s', T, \lambda)$

$s = s'$;

Actualizar a temperatura T;

Algoritmo 3.2 *Pareto Simulated Annealing*

3.2.3 Pesquisa Tabu

Em [Gandibleux et al. 1996] é proposta uma meta-heurística multiobjectivo baseada em TS, designada *"Multiple Objective Tabu Search"* (MOTS). Para a selecção de uma solução na vizinhança é usada uma função escalarizante ponderada, que considera um ponto ideal de âmbito local, com o melhor valor em cada objectivo de todas as soluções da vizinhança. A abordagem utiliza duas listas tabu: uma de atributos e outra de pesos. O vector de pesos é alterado periodicamente, com degradação dos pesos nos objectivos melhorados.

Em [Hansen 1997] e [Hansen 1998] é proposta uma outra meta-heurística multiobjectivo baseada em TS, que, para diferenciação da anterior, é referida com a

abreviatura MOTS*. À semelhança do PSA, a MOTS* actua sobre uma população de soluções, com o objectivo de construir uma aproximação da fronteira de Pareto. Os mecanismos base utilizados são:

1. *Determinação para cada solução de uma direcção de optimização antes da selecção e realização do movimento.* Esta direcção de optimização é dada por um vector de pesos associado a cada solução e é caracterizada por:
   - uma orientação para a fronteira de Pareto, garantida pela utilização de pesos não-negativos;
   - um afastamento das restantes soluções da população não-dominadas pela solução considerada, afastamento este mais forte em relação às soluções mais próximas.
2. *Realização de "drift".* Verifica-se que as soluções encontradas têm tendência a concentrar-se em determinadas regiões, pelo que se torna necessário um mecanismo que condicione a sua movimentação. O mecanismo base utilizado consiste na substituição de uma solução por uma cópia de outra. Uma possível sofisticação deste procedimento consistirá no redimensionamento da população em função de uma contagem para cada solução do número de soluções que a dominam. Se, em média, esse número for muito elevado, haverá demasiadas soluções na população e pode-se reduzir o seu número. Se esse número for reduzido, haverá poucas soluções e o seu número pode ser aumentado.

Algoritmo *MOTS**

Calcular um conjunto de soluções de partida admissíveis $G \subset S$;

Inicializar o vector de *"range equalization factors"* ð com $p_k = 1/k$;

Inicializar a aproximação ao conjunto eficiente $E = \{\}$;

Para cada $s_i \in G$

Inicializar a respectiva lista tabu $TL_i = \{\}$;

Actualizar E com s;

Enquanto não se cumprir o critério de paragem

Para cada $s_i \in G$

Inicializar o respectivo vector de pesos $ë_i = 0$;

Para cada $t \in G$ **tal que** $f(t)$ não dominada por $f(s_i)$ **e** $f(t) \neq f(s_i)$

Calcular a proximidade $w = g(d(f(s_i), f(t), \delta))$

Para cada objectivo j **tal que** $f_j(s_i) < f_j(t)$

$l_{ij} = l_{ij} + p_j \cdot w$;

Se $\lambda_i = 0$

Atribuir aos elementos do vector de pesos λ_i associado a s_i

valores aleatórios tais que $\lambda_{ij} \geq 0, \forall j \; e \sum_{j=1}^{k} \lambda_j = 1$;

Determinar $s' \in N(s)$ com $\lambda \times f(s') \leq \lambda \times f(s'')$, para todo o $s'' \in N(s)$,

e (TL_i não faz (s_i, s') tabu **ou** s' satisfaz o critério de aspiração);

Inserir atributos de (s_i, s') em TL_i, removendo o primeiro elemento se

TL_i estiver completa;

$s_i = s'$;

Actualizar E com s';

Actualizar δ;

Se se verificar o critério de *"drift"*

Substituir uma solução de G seleccionada aleatoriamente por

uma outra solução de G seleccionada aleatoriamente.

Algoritmo 3.3 *MOTS**

Para tomar em consideração as ordens de grandeza dos diversos objectivos, os respectivos valores deverão ser corrigidos através da multiplicação por *factores de equalização de gamas* (*"range equalization factors"*), conforme sugerido em [Steuer 1986]:

$$p_i = \frac{1}{Range_i} \left[\sum_{i=1}^{k} \frac{1}{Range_i} \right]^{-1}, \tag{3.17}$$

$$Range_i = \max_{s \in E} f_i(s) - \min_{s \in E} f_i(s),$$

Para a definição das gamas (*"ranges"*) é possível, na ausência de outra fonte de conhecimento, usar as gamas das soluções do conjunto E, motivo pelo qual se deve realizar a sua actualização, sempre que há actualização da aproximação ao conjunto eficiente.

A proximidade deverá ser uma função g da distância, decrescente e com valores positivos. A função de distância deverá ser baseada numa métrica do espaço de objectivos, usando factores de equalização de gamas. A função $g(d) = 1/d$ tem sido usada com bons resultados. Para a medida de distâncias é sugerida a *distância de Manhattan*

$$d(s, s', \delta) = \sum_{i=1}^{k} p_i |s_i - s'_i| . \tag{3.18}$$

O critério de aspiração definido consiste na aceitação de qualquer solução não-dominada.

Em [Hansen 1997] são sugeridas algumas extensões possíveis a esta versão básica do algoritmo, das quais destacamos os seguintes aspectos de âmbito mais geral:

1. *Realização de "drift" com ajuste do número de soluções actuais.* O número de soluções é um parâmetro difícil de definir: um número reduzido poderá não cobrir razoavelmente a fronteira de Pareto, mas demasiadas soluções podem gerar excessiva sobreposição, com oscilações exageradas, exigir demasiados cálculos e retardar a convergência para a fronteira de Pareto. Isto dependerá evidentemente da forma da fronteira e da vizinhança usada.

   O grau de dominância de uma solução, dado pelo número de outras soluções que a dominam, constitui um bom indicador de sobreposição, a utilizar para determinar o ajuste do número de soluções.

2. *Limiares de indiferença.* É sugerida uma regra geral para a inserção de uma solução na aproximação ao conjunto eficiente E: se a nova solução domina uma ou mais das soluções de E, a solução é inserida (removendo-se outras e não aumentando o tamanho do conjunto); se não, a solução é inserida apenas se ultrapassar um limiar de diferença em relação às restantes soluções.

3. Para *moderar a flutuação dos pesos*, pode ser utilizada uma média móvel sobre os vectores de pesos de cada uma das soluções.

4. Pelo facto de utilizar um *programa de soma ponderada*, a versão básica do algoritmo poderá não ser eficiente na localização de soluções não-suportadas. Em muitos casos tal não constituirá um problema, porque também soluções não-suportadas são inseridas em E, mas o procedimento poderá ser mais correcto usando um programa de Tchebycheff ponderado aumentado.

5. O potencial interesse de utilizar *outras técnicas* próprias de TS com um só objectivo, em especial implementando oscilação estratégica e memória de longo prazo.

3.2.4 Contexto de apoio à decisão

Conforme é referido em [Coello 1999], embora os processos de tomada de decisão *a priori* e *a posteriori* sejam comuns na literatura de Investigação Operacional, as abordagens interactivas têm sido normalmente favorecidas pelos investigadores [Gardiner, Steuer 1994]. A integração de qualquer procedimento para resolução de Problemas de Optimização Combinatória Multiobjectivo, incluindo meta-heurísticas multiobjectivo, num processo de tomada de decisão é condicionada pela forma como o agente de decisão é, ou pode ser, envolvido nesse processo [Viana 1997]:

1. Existindo disponibilidade de informação *a priori*, poderá ser viável a definição de uma função de avaliação única, e a redução do problema a um problema de optimização com um só objectivo ou a uma série de problemas de optimização com um só objectivo.

2. Quando o agente de decisão apenas disponibiliza informação *a posteriori*, será necessário usar métodos que permitam determinar o conjunto de soluções eficientes, a partir do qual será seleccionada uma solução de compromisso.

3. Se o processo de optimização e a tomada de decisão forem articulados de forma interactiva, a cada passo o agente de decisão recebe informação do processo de optimização, que avalia e, mediante essa avaliação, fornece informação suplementar que condiciona a evolução desse processo.

O exemplo clássico do primeiro caso é a utilização de funções de agregação, em que os pesos são definidos à partida, por forma a combinar todos os objectivos numa única função objectivo. O segundo caso, no qual se tem inscrito a maioria das meta-heurísticas multiobjectivo, consiste em realizar em primeiro lugar a pesquisa de soluções e, em segundo, a tomada de decisão. Neste caso, poderá haver lugar a um processo interactivo, mas de selecção de soluções entre as soluções da aproximação ao conjunto eficiente. Este processo é interessante nas situações, bastante frequentes, em que existe um grande número de soluções na aproximação ao conjunto eficiente. Em [Coello 1999] e [Hansen 1997] podem ser recolhidas referências de algumas abordagens que se enquadram neste caso. Relativamente ao terceiro caso, em [Coello 1999] é apresentada apenas uma abordagem de GA que integra a articulação interactiva de preferências por parte do agente de decisão [Fonseca, Fleming 1993]: em cada geração, o agente de decisão estabelece metas para os diversos objectivos, reduzindo o

conjunto de soluções que inspecciona e aprendendo, dessa forma, acerca dos compromissos entre objectivos.

Em [Hansen 1997] é proposta a utilização interactiva de MOTS*, de uma forma extensível ao PSA, com base na geração parcial do conjunto eficiente através da definição de direcções de pesquisa, que podem ser combinadas com os vectores de pesos. Estas direcções podem resultar directamente de informação disponibilizada pelo decisor ou indirectamente, no caso de pontos de referência, através da criação de vectores de direcção entre as soluções actuais e esses pontos. A solução apresentada no trabalho referido permite controlar, através de um parâmetro, a intensificação da pesquisa na direcção de referência. Este parâmetro poderá começar por ter um valor baixo e ir aumentando à medida que o decisor vai conhecendo o "espaço de compromissos" do problema, e articulando as suas preferências. À medida que esse conhecimento é adquirido, pode também ser útil a alteração dos níveis mínimos (que devem ser atingidos para uma solução ser considerada) e de saturação (além dos quais não é necessária mais optimização), facilmente incorporáveis nos métodos referidos.

3.3 Flexibilização em meta-heurísticas multiobjectivo

No Capítulo 2 referiram-se três tendências de flexibilização em meta-heurísticas que serão objecto de especial atenção neste trabalho. O trabalho de investigação sobre meta-heurísticas multiobjectivo, em particular no domínio da pesquisa local, é ainda exíguo, mas, apesar disso, é já possível encontrar algumas referências às potencialidades que esses aspectos de flexibilização poderão conferir às soluções algorítmicas e à sua implementação. Essas referências, conjuntamente com a relevância das tendências de flexibilização em meta-heurísticas para um só objectivo, são as principais razões para a sua aplicação no âmbito das meta-heurísticas multiobjectivo.

3.3.1 Pesquisa local multiobjectivo

O presente trabalho debruça-se, particularmente, sobre meta-heurísticas baseadas em pesquisa local aplicadas a Problemas de Optimização Combinatória Multiobjectivo. Neste contexto, é colocado um ênfase especial nas abordagens baseadas na noção de óptimo de Pareto, designadamente PSA e MOTS*.

Conforme referido aquando da introdução da MOTS*, ambas as abordagens trabalham sobre uma população de soluções, procurando construir uma aproximação da fronteira de Pareto. O mecanismo utilizado para definir direcções de optimização para cada uma das soluções baseia-se, também nas duas abordagens, na utilização de vectores de pesos associados a cada uma das soluções. Estes pesos são definidos de forma distinta em cada uma das abordagens, mas com fins idênticos: orientar a pesquisa para a fronteira de Pareto e procurar que as soluções se dispersem o mais possível sobre a mesma. O primeiro objectivo é assegurado com pesos não-negativos. O segundo através da comparação das soluções em análise com outras soluções do conjunto de soluções actuais. Embora de formas distintas, ambos os métodos (PSA e MOTS*) actuam sobre cada solução actual, pesquisando e seleccionando uma solução na respectiva vizinhança para substituir essa solução actual.

A identificação destes aspectos comuns sugere a definição de um *template* de Pesquisa Local Multiobjectivo.

A consideração de um *contexto* para cada solução tem como objectivo enquadrar os diversos componentes de meta-heurísticas dependentes das soluções individuais, como é, por exemplo, o caso das listas tabu na MOTS*.

A activação de estratégias complementares permite acomodar elementos como a realização de *"drift"* ou o ajuste do número de soluções da população.

Embora não considerado explicitamente no PSA, um aspecto geral que pode ser identificado como comum é a utilização de factores de equalização de gamas, inicializados a partir da aproximação inicial ao conjunto de soluções não-dominadas, e actualizados sempre que se verificar alguma alteração sobre esse conjunto.

A definição deste *template* poderá, à semelhança do trabalho de [Vaessens et al. 1995] vir a permitir classificar os vários algoritmos existentes, identificar novas abordagens algorítmicas e sugerir variantes inovadoras.

No presente trabalho, o *template* orientará o desenvolvimento de uma infra-estrutura algorítmica comum às diversas abordagens de pesquisa local multiobjectivo, constituirá uma base para a integração de estratégias de carácter mais geral, e sugerirá uma abordagem de hibridização e paralelização.

Template *Pesquisa Local Multiobjectivo*

Calcular um conjunto de soluções de partida admissíveis $G \subset S$;

Inicializar a aproximação ao conjunto eficiente $E = \{\}$;

Para cada $s_i \in G$

Inicializar o respectivo contexto;

Actualizar E com s;

Inicializar o vector de factores de equalização de gamas δ;

Enquanto não se cumprir o critério de paragem

Para cada $s_i \in G$

Actualizar o respectivo vector de pesos λ_i;

Inicializar a solução seleccionada $s_s = \varnothing$;

Para cada $s' \in N(s)$

Actualizar E com s';

Actualizar δ;

Se s' seleccionável e s' preferível a s_s

$s_s = s'$;

Se $s_s \neq \varnothing$ e s_s aceitável

$s_i = s_s$;

Actualizar o respectivo contexto;

Se se verificar o respectivo critério

Activar alguma estratégia complementar;

Algoritmo 3.4 *Pesquisa Local Multiobjectivo*

3.3.2 Vizinhanças variáveis e estratégias de listas de candidatos

Em [Hansen 1997] é referido o interesse especial, no contexto da MOTS*, do uso de técnicas adicionais, originariamente utilizadas em TS com um só objectivo. O mesmo autor, em [Hansen 1997b], incorpora uma técnica de TS reactiva, o uso de *"hashing vectors"*, no

contexto da MOTS*. Esta técnica tem como objectivo impedir a ocorrência de ciclos, libertando a lista tabu para outras tarefas, designadamente a definição de uma trajectória de pesquisa no espaço de soluções.

No presente trabalho procurar-se-á avaliar o interesse da incorporação em meta-heurísticas multiobjectivo, de estratégias de carácter mais geral, nomeadamente as introduzidas no Capítulo 2: vizinhanças variáveis e estratégias de listas de candidatos.

A utilização destas abordagens integra-se de forma perfeitamente natural no *template* de pesquisa local multiobjectivo, devendo considerar-se as seguintes particularidades:

- cada solução deverá ter as suas próprias estrutura de vizinhança e lista de candidatos, o que poderá ser assegurado com a consideração de uma réplica destes componentes no contexto de cada uma das soluções;
- de acordo com o referido no Capítulo 2, a respeito do SA, a aplicação de listas de candidatos com PSA exige que se considere, em cada iteração, a pesquisa de soluções candidatas numa sub-vizinhança de dimensão superior à unidade;
- no contexto da MOTS*, utiliza-se a função escalarizante *soma ponderada* para a selecção de uma solução na vizinhança da solução actual; este mesmo mecanismo poderá ser usado com PSA, uma vez que também aqui são considerados conjuntos de pesos.

3.3.3 Hibridização e paralelização

Em [Czyzak, Jaskiewicz 1998] é referida a natural aptidão destas abordagens para paralelização, uma vez que se baseiam no processamento de populações de soluções e que todo o conjunto de operações para uma solução (definição de pesos, construção de nova solução, aceitação da nova solução) pode ser feito em diferentes processadores. Um caso paradigmático é o dos GA, que foram pioneiros quer na aplicação de meta-heurísticas num contexto multiobjectivo, quer na utilização com sucesso de abordagens de paralelização.

No caso do *template* de pesquisa local multiobjectivo apresentado, o processamento associado a cada uma das soluções pode ser inteiramente executado em processadores separados. A própria realização de *"drift"* no algoritmo MOTS* pode ser facilmente modificada de forma a permitir a sua paralelização. As únicas estruturas de dados que deverão ser de acesso comum aos diversos processos são a população de soluções actuais, o conjunto de soluções eficientes e os factores de equalização de gamas.

Esta estrutura de paralelização indicia já uma abordagem natural de hibridização, que consiste em ter várias meta-heurísticas PSA e MOTS* a trabalhar uma população comum de soluções. A abordagem geral proposta a este nível será exactamente a de considerar um vector

de meta-heurísticas que, por motivos de simplicidade de coordenação, trabalham sobre as suas próprias sub-populações, mas consideram uma população global para a definição dos pesos e um conjunto de soluções eficientes e factores de equalização de gamas comuns. Esta abordagem será implementada com o lançamento de vários *"threads"* paralelos correspondentes às várias meta-heurísticas, que cooperarão através de uma população e conjunto de soluções eficientes comuns. Tratar-se-á, consequentemente, de uma abordagem de paralelização do tipo 3 (*"threads"* multipesquisa), cooperativa e de hibridização de alto nível, co-evolutiva. Em ambas as perspectivas, paralelização e hibridização, a abordagem adoptada parece ter um considerável potencial.

3.4 Conclusões

A optimização em contextos multiobjectivo tem vindo a ser objecto de uma atenção crescente. Num esforço que, certamente, tem sido fortemente motivado pelo sucesso das meta-heurísticas em contextos de objectivo único, estas têm vindo a ser adaptadas para tratar objectivos múltiplos. Tratando-se de uma linha de investigação recente, a maioria das meta-heurísticas multiobjectivo é baseada em GA, SA e TS, sendo de esperar, naturalmente, a extensão destas abordagens a outras meta-heurísticas.

Os GA desempenharam um papel pioneiro nesta área, adaptando-se especialmente bem à utilização em contextos multiobjectivo essencialmente por dois motivos: ao trabalhar com uma população de soluções, podem facilmente procurar em paralelo as múltiplas soluções do conjunto eficiente, eventualmente explorando semelhanças entre as soluções; são, também, menos sensíveis às questões de forma e continuidade do conjunto de soluções não-dominadas do que as técnicas tradicionais de programação matemática.

Esta segunda característica é partilhada com o conjunto de abordagens baseadas em SA e TS. Um grupo destas abordagens baseia-se na repetição de execuções de meta-heurísticas com um único objectivo, sendo os objectivos combinados numa única função agregadora, normalmente uma função escalarizante *soma ponderada*. Os pesos nesta função estabelecem uma direcção de pesquisa, cuja variação em cada execução tem como fim permitir uma aproximação completa do conjunto de soluções não-dominadas. Num outro grupo, que inclui PSA e MOTS*, a primeira característica é introduzida em abordagens baseadas em SA e TS, através da consideração de uma população de soluções, cada uma das quais possui um conjunto próprio de pesos. Estes pesos são calculados dinamicamente de forma a que cada solução se mova em direcção à fronteira não-dominada e se distancie das restantes soluções da população não-dominadas relativamente a ela.

Este trabalho debruça-se em particular sobre o segundo grupo de abordagens baseadas em SA e TS. Com base na identificação do comportamento comum a essas

abordagens, sugere-se um *template* de pesquisa local multiobjectivo, que desempenhará um papel central no *framework* desenvolvido. Propõe-se, ainda, a integração (natural) neste *template* de estratégias de carácter mais geral, nomeadamente vizinhanças variáveis e estratégias de listas de candidatos. Por fim, partindo da observação das possibilidades de paralelização do *template*, sugere-se uma abordagem de paralelização e hibridização de alto nível. Estas propostas enquadram-se no conjunto de tendências de flexibilização em meta-heurísticas apresentadas no Capítulo 2, e apresentam-se, portanto, particularmente interessantes para a aplicação de abordagens orientadas por objectos.

4
ABORDAGENS ORIENTADAS POR OBJECTOS PARA META-HEURÍSTICAS

Neste capítulo apresenta-se um conjunto de abordagens orientadas por objectos para meta-heurísticas, que têm sido objecto de um interesse crescente, essencialmente motivado pela necessidade de aproximar, nesta área, a teoria e a aplicação, e de criar formas mais eficientes de implementação e comparação de métodos.

De facto, o trabalho descrito nesta dissertação foi, também, realizado na sequência das duas ordens de motivações acima referidas. Embora neste contexto se tenha essencialmente valorizado a segunda perspectiva, o objectivo último é a posterior utilização dos resultados obtidos em Sistemas de Apoio à Decisão (SAD) baseados em meta-heurísticas multiobjectivo.

Na secção 4.1 introduzem-se os conceitos básicos de Orientação por Objectos, complementados nas secções 4.2 e 4.3 com a apresentação de duas abordagens que actualmente assumem grande relevância no desenvolvimento com base neste paradigma: *padrões de desenho* (*"design patterns"*) e *frameworks*. Na secção 4.4 procede-se à apresentação das diversas abordagens orientadas por objectos para meta-heurísticas descritas na literatura, e duas outras abordagens com aspectos relevantes neste contexto - um *template* e uma linguagem. Por fim, apresenta-se um pequeno conjunto de conclusões.

4.1 Conceitos básicos de Orientação por Objectos

Para uma introdução sintética dos principais conceitos de Orientação por Objectos (OO), seguir-se-á de muito perto [Russo 1991], um dos trabalhos pioneiros na área dos *frameworks*, que inclui uma excelente apresentação desses conceitos. Refere-se ainda [Booch 1994] para uma descrição mais pormenorizada desta matéria.

Os quatro princípios fundamentais do paradigma OO, que caracterizam todas as abordagens nele baseadas, são: o encapsulamento de dados, a abstracção de dados, o polimorfismo e a herança.

4.1.1 Encapsulamento de dados

O principal objectivo da utilização de técnicas de encapsulamento de dados é o aumento da modularidade, da facilidade de manutenção e da fiabilidade do *software*. Tais técnicas têm como princípios básicos a agregação dos dados com as funções que sobre elas operam e a restrição do acesso aos dados a apenas essas funções. A preservação de estado entre chamadas de funções é garantida através do seu armazenamento nos dados encapsulados.

No paradigma OO, a unidade de encapsulamento de dados é o *objecto*. Um objecto é um encapsulamento simples, consistindo de um conjunto de variáveis e um conjunto de operações que permitem aceder a essas variáveis e alterá-las.

Uma operação é uma *mensagem* que o objecto *aceita*. Invocar uma operação de um objecto consiste no *envio de uma mensagem* ao objecto. O conjunto de mensagens que um objecto aceita é a sua *assinatura* ou *protocolo*.

Em OO, o *envio de mensagem* existe apenas numa perspectiva conceptual: o que realmente se realiza é a pesquisa (*"lookup"*) de um *método* (o código que executa a operação descrita pela mensagem), com base no respectivo objecto, e a sua invocação. Enviar uma mensagem a um objecto resulta, portanto, numa *pesquisa de método* para encontrar o método em causa, seguido de uma *invocação de método* para o chamar. Idealmente, os métodos de um objecto constituem a única forma de outros objectos do sistema terem acesso ao seu estado e realizarem operações sobre ele.

As mensagens que um objecto aceita podem ser *públicas* ou não. Todas as mensagens classificadas como públicas são aceites pelo objecto, independentemente da sua origem. O conjunto de mensagens públicas da assinatura do objecto é o seu *interface*.

À semelhança de outras técnicas de encapsulamento de dados, os objectos preservam o seu estado entre sucessivos envios de mensagens, recorrendo às respectivas variáveis.

4.1.2 Abstracção de dados

Os programas de computador podem ser vistos como modelos, que utilizam, enquanto tal, abstracções de entidades, e nos quais os valores manipulados pelas expressões correspondem às entidades modeladas. No paradigma OO todos os valores são objectos. Uma vez que muitos objectos representam abstracções semelhantes e, consequentemente, partilham comportamentos idênticos, o paradigma OO introduz o conceito de *classe*, por forma a permitir exprimir esses aspectos comuns.

Uma classe é um padrão para a criação de um tipo de objecto: define as variáveis, a assinatura e a implementação das mensagens da assinatura para as *instanciações* ou *instâncias* da classe. Os métodos de uma classe podem referenciar as variáveis que a classe define. Em execução (*"run-time"*), estas variáveis são ligadas às de uma instância particular da classe. Esta ligação em execução (*"run-time binding"*) é também um factor distintivo do paradigma OO.

Num programa, as expressões geram valores que são normalmente atribuídos a variáveis ou usados noutras expressões. A *verificação de tipo* (*"type checking"*) é o mecanismo usado para determinar se esses valores são usados no contexto apropriado. Existem duas formas de verificação de tipo: *estática* e *dinâmica*.

As linguagens de *tipos estáticos* (*"statically typed"*) exigem a determinação, durante a compilação, da conformidade entre valores e tipos requeridos nos pontos de utilização. Em OO, estes tipos são especificados como assinaturas que um objecto atribuído à variável deve possuir. Em muitos casos, a assinatura é especificada como uma classe, cuja assinatura implicitamente define o tipo da variável. Em alguns casos, a especificação de assinaturas pode ser independente de classes.

Em linguagens de *tipos dinâmicos* (*"dinamically typed"*), a conformidade entre um valor e um tipo particular exigido é verificada em execução. Em OO, tal exige que as variáveis não tenham tipo e possam referenciar objectos de qualquer tipo.

Estas duas formas de verificação de tipo apresentam diferentes compromissos aos níveis de engenharia de *software* e de desempenho: as abordagens dinâmicas conduzem habitualmente a código mais flexível e de mais simples reutilização; no entanto, são em geral menos eficientes, uma vez que não existe informação de tipo, para orientar a compilação.

4.1.3 Herança e subclasses

Da mesma forma que as classes surgem naturalmente de comportamentos idênticos de grupos de objectos, diferentes classes podem ter mensagens e métodos comuns. A *herança de classes* permite a um conjunto de classes partilhar partes de um interface comum (assinatura) e partes de uma implementação comum (métodos e variáveis).

A *derivação* de *subclasses* é o mecanismo de herança mais conhecido, que permite que uma parte ou a totalidade de assinatura, métodos e variáveis de uma classe sejam herdados de *classes base*. A herança simples permite apenas uma classe base, enquanto a herança múltipla permite diversas classes base. As relações de herança entre classes formam grafos acíclicos não-orientados, pelo que são normalmente designadas *hierarquias de classes*. As classes descendentes de uma dada classe numa hierarquia são normalmente designadas as suas *subclasses* e dizem-se *derivadas* dessa classe.

A *delegação*, em que os objectos encaminham mensagens para outros objectos, é uma outra forma de herança de comportamento.

A herança permite realizar adaptações de classes de uma forma incremental e estruturada. Para implementar uma nova classe é possível ampliar a assinatura herdada com mensagens adicionais e/ou redefinir a implementação de métodos herdados. As classes que apenas definem uma assinatura e deixam a definição da sua implementação para outras classes são normalmente designadas *classes abstractas*. As classes que definem uma implementação para uma assinatura particular são designadas *classes concretas*. Na prática, muitas classes abstractas encontram-se algures entre o puramente abstracto e o concreto, frequentemente disponibilizando métodos para algumas, mas não todas, as mensagens nas suas assinaturas.

O mecanismo que permite a estruturação em classes abstractas e concretas é o diferimento da ligação (*"delayed binding"*) das mensagens com os métodos que as implementam: quando uma mensagem é enviada a um objecto, o método concreto a invocar só é determinado em execução, através da classe do objecto.

A separação de interface e implementação, permitida pelo uso de classes abstractas e concretas, é importante para a portabilidade. As classes abstractas podem definir interfaces que são implementados por diferentes classes concretas em diferentes aplicações. A separação torna também mais fácil a compreensão do sistema, uma vez que as abstracções podem ser concebidas de forma separada da sua implementação.

4.1.4 Polimorfismo

Um método que pode aceitar argumentos de tipos diferentes e, de acordo com eles, comportar-se de forma distinta, é dito *polimorfo em relação a* esses argumentos. Em OO, tal é conseguido pelo mecanismo já descrito de diferimento da associação das mensagens com os métodos que as implementam. Este tipo de polimorfismo, em que os objectos determinam o método apropriado em execução é chamado *polimorfismo dinâmico*. Por outro lado, constitui também uma forma de *polimorfismo de inclusão* ou *limitado*, que restringe o polimorfismo a objectos que partilham uma representação ou assinatura comum. O polimorfismo de inclusão *baseado em assinatura* é prevalecente em linguagens OO.

Cada parâmetro formal de um método tem uma assinatura implícita, *S*, que consiste no conjunto de mensagens que serão enviadas ao parâmetro dentro do método e de qualquer método ao qual o argumento seja passado, a partir daí. O argumento usado para satisfazer um parâmetro polimorfo deve ser capaz de aceitar as mensagens definidas pela assinatura *S* desse parâmetro.

As linguagens de tipos estáticos que especificam tipos de parâmetros formais como assinaturas podem implementar polimorfismo na sua forma mais essencial, através da comparação de assinaturas, em compilação, para verificar a conformidade do parâmetro concreto com o tipo especificado para os parâmetros formais. As linguagens de tipos estáticos que especificam tipos de parâmetros indirectamente com uma classe, implementam uma forma restrita de polimorfismo: apenas instâncias de uma classe específica, ou suas subclasses, constituem argumentos aceitáveis, mesmo que instâncias de outras classes possam ter a assinatura desejada. Este comportamento é designado *polimorfismo de herança*.

As linguagens de tipos dinâmicos implementam polimorfismo na sua forma mais essencial, uma vez que qualquer objecto argumento só será alvo de verificação de tipo em execução.

Uma outra forma de polimorfismo é o *polimorfismo estático*, ou *sobrecarga de operadores*, no qual o compilador selecciona o método com base nos tipos dos parâmetros. Esta é a forma clássica de polimorfismo que ocorre nas linguagens de programação tradicionais (um exemplo é a forma como o operador + opera sobre argumentos inteiro/inteiro, real/real, inteiro/real ou real/inteiro).

O polimorfismo permite um elevado grau de flexibilidade na concepção e reconfiguração de *software* OO e é crítico para o desenho de código reutilizável: permite acrescentar novos componentes e substituir componentes existentes, uma vez que novas classes

que implementam assinaturas já existentes podem ser usadas com o código que pressupõe essas assinaturas.

4.2 *Padrões de desenho ("Design Patterns")*

4.2.1 Definições e conceitos fundamentais

A origem do conceito de *padrões* (*"patterns"*) situa-se no trabalho do arquitecto Cristopher Alexander, que desenvolveu a ideia de uma *linguagem de padrões* (*"pattern language"*) para projectar casas e comunidades [Alexander 1977]. Na área de desenvolvimento de *software* segundo o paradigma OO, o conceito encontra aplicação na resolução de problemas elementares e estruturados de desenvolvimento.

À semelhança do trabalho de Alexander, em que diferentes *padrões* são propostos para diferentes fases de um projecto de arquitectura, também para diferentes fases do desenvolvimento de *software* - análise e desenho (*"design"*) - têm sido propostos diferentes *padrões*. Em [Coad 1992] foram introduzidos os *padrões de análise* (*"analysis patterns"*), que resultam da modelação de problemas em domínios aplicacionais específicos. Em [Buschmann, Meunier 1994] encontra-se uma caracterização de diversas classes de *padrões* da área de desenho, nos quais o ênfase é colocado na reutilização de soluções de desenho.

Uma outra perspectiva global classifica os *padrões* em *generativos* e *não-generativos*. Os primeiros são activos e prescritivos, incidindo sobre o processo de construção de uma solução. Os segundos são passivos e descritivos, surgem da observação de sistemas já construídos e descrevem fundamentalmente relações entre classes e/ou objectos.

No presente trabalho a atenção será dedicada aos *padrões de desenho (OO)*, uma classe de *padrões* não-generativos, específicos para a fase de desenho de *software*, aos quais tem sido dedicada especial atenção na literatura e também na prática de desenvolvimento de *software*.

Em [Gamma et al. 1995] os *padrões de desenho* são definidos como estruturas que captam e exprimem experiência de desenho, identificando, denominando e abstraindo situações frequentes em desenho OO. Permitem preservar informação de desenho, captando o seu propósito e identificando classes e instâncias, os seus papéis, colaborações e distribuição de responsabilidades. Constituem soluções de arquitectura de *software* de escala reduzida, que envolvem apenas algumas classes, podendo ser vistos como blocos de construção imediatamente acima das classes concretas e abstractas.

Um *padrão de desenho* consiste de quatro partes essenciais [Gamma et al. 1995]:

1. o *nome do padrão*;
2. o *problema* de desenho OO para o qual o *padrão* é apropriado;
3. a *solução* que o *padrão* constitui, descrevendo os elementos que a constituem, as suas relações, responsabilidades e colaborações;
4. as *consequências* da aplicação do *padrão*, ou seja os resultados e compromissos inerentes à sua aplicação.

4.2.2 Descrição de *padrões de desenho*

Para a descrição de *padrões de desenho*, diversos autores têm proposto a utilização de *templates*, que visam facilitar a sua compreensão, comparação e utilização. Nestes *templates* são contemplados aspectos que permitam, da forma mais completa possível, disponibilizar o resultado final do processo de desenho, bem como as decisões, alternativas e compromissos que a este conduzem, e exemplos concretos de aplicação.

Dois dos *templates* mais divulgados são os propostos em [Gamma et al. 1995] e [Buschmann, Meunier 1994].

O *template* de [Gamma et al. 1995] engloba os seguintes aspectos:

- *Nome e classificação do padrão*.
- *Intenção*: um resumo do que faz o *padrão*, da sua lógica e propósitos, e dos aspectos ou problemas de desenho particulares por ele abordados.
- *Também conhecido como*: outros nomes bem conhecidos do *padrão*.
- *Motivação*: um cenário ilustrando um problema de desenho e a forma como as estruturas de classes e objectos do *padrão* o resolvem.
- *Aplicabilidade*: situações de aplicação do *padrão* e exemplos de insuficiências de desenho que a sua aplicação permite cobrir.
- *Estrutura*: uma representação gráfica das classes do *padrão*, usando uma notação baseada em *"Object Modeling Technique"* (OMT) e *Diagramas de Interacção* para ilustrar sequências de pedidos e colaborações entre objectos.
- *Participantes*: as classes e/ou objectos participantes no *padrão* e respectivas responsabilidades.
- *Colaborações*: a forma como as classes e/ou objectos participantes no *padrão* colaboram para desempenhar as suas responsabilidades.
- *Consequências*: a forma como o *padrão* suporta os seus objectivos, os compromissos e resultados da sua utilização e os aspectos da estrutura do sistema que podem variar de forma independente.

- *Implementação*: problemas, sugestões ou técnicas a considerar ao implementar o *padrão*.
- *Código exemplo*: fragmentos de código que ilustram formas de implementação do *padrão*.
- *Usos conhecidos*: exemplos do *padrão* encontrados em sistemas reais.
- *Padrões relacionados*: relações próximas com outros *padrões* e diferenças importantes.

O *template* de [Buschmann, Meunier 1994] é praticamente idêntico a este, mas com três secções adicionais:

- *Metodologia*: enumeração dos passos metodológicos para construir o *padrão*.
- *Variantes*: descrição das possíveis variantes do *padrão*.
- *Comportamento dinâmico*: ilustração do comportamento dinâmico do *padrão*.

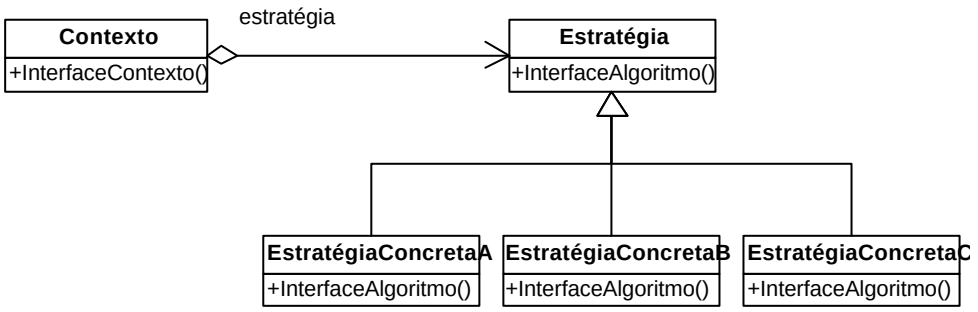
A classificação de *padrões de desenho* constitui um dos aspectos essenciais da sua descrição. Por um lado, facilita a sua compreensão e selecção, face ao número tipicamente elevado de *padrões* apresentados em catálogos; por outro, os esquemas de classificação podem também orientar a detecção de novos *padrões*. Os esquemas de classificação propostos têm como base diversas perspectivas sobre os *padrões de desenho*, apresentando-se de seguida alguns dos mais destacados na literatura:

- O *tipo de erro de reutilização* [Gamma 1993] que o *padrão* cobre. Em geral estes erros estão relacionados com dependências em relação a aspectos particulares, como, por exemplo, detalhes específicos de implementação ou representação.
- *Funcionalidade* e *princípio estrutural do padrão* [Buschmann, Meunier 1994]. Na funcionalidade inclui-se a criação de objectos, o acesso a objectos, a gestão da comunicação entre objectos e a organização da realização de tarefas complexas. Ao nível dos princípios estruturais encontram-se a abstracção, o encapsulamento, a separação de aspectos e a coesão.
- *Camadas* de *padrões*, que podem incluir *padrões* de camadas subjacentes [Zimmer 1994]: *padrões* básicos; *padrões* dedicados a problemas típicos de desenho, que podem integrar *padrões* básicos; *padrões* específicos a domínios aplicacionais, que podem integrar *padrões* das duas camadas anteriores.
- *Âmbito* (*"scope"*) e *propósito* (*"purpose"*) [Gamma et al 1995]. O âmbito é relativo à predominância de relações entre classes e suas subclasses ou entre objectos, enquanto o propósito reflecte o que o *padrão* faz, com distinção entre *padrões* criadores (*"creational"*), estruturantes (*"structural"*) e comportamentais (*"behavioural"*).

4.2.3 Um exemplo: o *padrão* Estratégia (*"Strategy"*)

O *padrão* Estratégia (*"Strategy"*) é um *padrão* comportamental, ao nível dos objectos. A título de exemplo, apresentam-se de seguida alguns extractos da descrição deste *padrão* [Gamma et al. 1995].

- *Intenção*: Definir uma família de algoritmos, encapsular cada um e torná-los intermutáveis. O *padrão* permite variar o algoritmo de forma independente do cliente que o usa.
- *Também conhecido como*: Política (*"Policy"*).
- *Motivação*: Existem muitos algoritmos para separar em linhas um texto editado. Implementar esses algoritmos nas classes que deles necessitam não é desejável por várias razões (...) Esses problemas podem ser evitados pela definição de classes que encapsulam diferentes algoritmos de separação por linhas. Um algoritmo que é encapsulado desta forma é uma **estratégia** (...)
- *Aplicabilidade*:
  - Muitas classes relacionadas diferem apenas no seu comportamento. As estratégias permitem configurar uma classe com um de muitos comportamentos.
  - Diferentes variantes de um algoritmo são necessárias, por exemplo, reflectindo diferentes compromissos espaço/tempo. As estratégias podem ser usadas quando estas variantes são implementadas como uma hierarquia de algoritmos.
  - Um algoritmo usa dados dos quais um cliente não deve ter conhecimento. O *padrão* pode ser usado para evitar expor estruturas de dados complexas e específicas do algoritmo.
  - Uma classe define muitos comportamentos, e estes aparecem como múltiplas condições nas operações da classe. As ramificações condicionais relacionadas podem ser agrupadas em classes de Estratégia individuais.
- *Estrutura*:



- *Participantes*:
  - **Estratégia**: declara um interface comum a todos os algoritmos suportados. O Contexto usa este interface para chamar o algoritmo definido por uma EstratégiaConcreta.
  - **EstratégiaConcreta**: implementa um algoritmo usando o interface de Estratégia.

- **Contexto**: é configurado com um objecto EstratégiaConcreta; mantém uma referência para um objecto Estratégia; pode definir um interface para permitir a Estratégia o acesso aos próprios dados.

- *Colaborações*:

  - Estratégia e Contexto interagem para implementar o algoritmo seleccionado. Um Contexto poderá passar todos os dados exigidos pelo algoritmo. Alternativamente, o Contexto poderá passar-se a si próprio como argumento em operações da Estratégia, o que permitirá a esta chamá-lo quando necessário.

  - Um Contexto transfere os pedidos dos seus Clientes à sua Estratégia. Os Clientes normalmente criam e passam um objecto EstratégiaConcreta ao Contexto, passando a interagir, então, exclusivamente com o Contexto. Existe frequentemente uma família de classes de EstratégiaConcreta da qual os clientes poderão escolher classes particulares.

- *Consequências*:

  - *Famílias de algoritmos relacionados.* Hierarquias de classes de Estratégia definem famílias de algoritmos ou comportamentos reutilizáveis por Contextos. A herança permite destacar funcionalidade comum aos algoritmos.

  - *Alternativa à derivação*. A derivação permite suportar uma variedade de algoritmos ou comportamentos: seria possível derivar uma classe Contexto directamente, para lhe conferir diferentes comportamentos. No entanto, desta forma, o comportamento fica rigidamente ligado ao Contexto e a implementação do algoritmo é misturada com a do Contexto, que será assim de compreensão, manutenção e extensão mais difíceis. A variação dinâmica de algoritmo é impossibilitada e a derivação resulta em muitas classes relacionadas, que diferem apenas no algoritmo ou comportamento que empregam. Encapsular o algoritmo em classes de Estratégia separadas permite variar o algoritmo independentemente do seu Contexto, tornando mais fácil a sua variação, compreensão e extensão.

  - *Eliminação de condições.* (...)

  - *Escolha de implementações.* As Estratégias podem disponibilizar diferentes implementações do mesmo comportamento. O Cliente pode escolher entre Estratégias com diferentes compromissos espaço-tempo.

  - *Necessidade de conhecimento das diferentes Estratégias*. O *padrão* tem uma desvantagem potencial no facto de que um Cliente deve compreender as diferenças entre as Estratégias antes de escolher a apropriada. Os Clientes podem, portanto, ser expostos a questões de implementação. O *padrão* deverá ser usado apenas quando a variação de comportamento é relevante para o Cliente.

  - *Overhead de comunicação entre Estratégia e Contexto*. O interface de Estratégia é partilhado por todas as classes EstratégiaConcreta, independentemente da complexidade dos algoritmos implementados. Assim, é provável que algumas dessas classes não utilizem toda a informação que lhes é passada através deste interface. (...)

  - *Aumento do número de objectos.* (...)

- *Implementação*: (...)

- *Código exemplo*: (...)

- *Usos conhecidos*: (...)

- *Padrões relacionados*: (...)

4.2.4 Aplicação de *padrões de desenho*

As experiências relatadas na literatura têm demonstrado a utilidade dos *padrões de desenho* como elementos de comunicação e têm confirmado também que os *padrões* contribuem para a qualidade do *software* e do processo de desenvolvimento [Mattsson 1996]. Essas experiências têm igualmente permitido ressaltar as principais desvantagens e aspectos críticos da sua utilização, de entre os quais se destacam alguns, apontados em [Mattsson 1996]:

- As possibilidades dos *padrões de desenho* são frequentemente sobrestimadas, ao contrário do esforço necessário para a sua implementação, que é muitas vezes subestimado. A sua utilização deve ocorrer num contexto de complementaridade com qualidades mais gerais de desenho e implementação, não da sua substituição.
- Os *padrões de desenho* surgem de experiências práticas, ou seja, são "descobertos" e não "inventados". Adicionalmente, existe a tentação de classificar qualquer novo "truque de programação" como um novo *padrão*: um *padrão* não deverá ser trivial e deverá ter tido mais do que uma aplicação.
- Os *padrões de desenho* não devem ser usados indiscriminadamente. O aumento de flexibilidade e a variabilidade que permitem são tipicamente conseguidos através da introdução de graus de liberdade adicionais, podendo, consequentemente, complicar um desenho. A sua aplicação deve ocorrer apenas quando a flexibilidade introduzida é realmente necessária. As consequências descritas num *padrão* apoiam de forma crucial esta decisão.

4.3 *Frameworks Orientados por Objectos*

4.3.1 Definições e conceitos fundamentais

A reutilização é um dos eixos de actuação mais importantes no esforço de aumento de produtividade no desenvolvimento de *software*. Para tal, o *software* deve ser projectado com preocupações específicas, tipicamente resultando em componentes de *software* gerais e extensíveis, nos quais se procura prever possíveis futuras aplicações e incorporar os correspondentes requisitos. Os *frameworks Orientados por Objectos* (que se designarão simplesmente por *frameworks*) surgem precisamente neste âmbito, cobrindo algumas lacunas deixadas em aberto por outras abordagens OO, igualmente suscitadas por preocupações na área da reutilização de *software*.

Existem, naturalmente, diversas definições para este conceito. Uma das mais conhecidas e divulgadas, de [Johnson, Foote 1988], considera um *framework* como um conjunto de classes que dá corpo a um desenho (*"design"*) abstracto de soluções para um conjunto de problemas relacionados. Os primeiros exemplos de *frameworks* situaram-se na área do interface com o utilizador: o ambiente do *Smalltalk*, *Model-View-Controller*, foi o primeiro *framework* largamente utilizado, enquanto a *Apple* desenvolveu o *framework MacApp* para o desenvolvimento de aplicações para o *Macintosh*.

Os *frameworks* distinguem-se das bibliotecas de classes por disponibilizarem comportamento por omissão e, consequentemente, uma maior facilidade de incorporação de conhecimento do domínio aplicacional. Em geral, as bibliotecas de classes contêm apenas pequenos componentes, a partir dos quais se pode construir uma aplicação, sendo necessário definir toda a comunicação entre esses componentes. No entanto, os *frameworks* não chegam a constituir aplicações, uma vez que apenas dão corpo à funcionalidade geral das aplicações de um determinado domínio, não cobrindo os aspectos que variam e são particulares aos casos de aplicação específicos. As aplicações constituem programas executáveis completos, que satisfazem uma determinada especificação de requisitos. Será ainda interessante contrastar os *frameworks* com os *padrões de desenho*: os *frameworks* são menos abstractos, incorporando, por exemplo, código; são, em geral, arquitecturas de maior dimensão (poderão conter *padrões de desenho*, não se verificando o simétrico); são mais especializados, situando-se sempre num domínio de aplicação específico.

Os elementos mais importantes no desenvolvimento de um *framework* são [Mattsson 1996]:

- As *classes abstractas*, que são fundamentais do ponto de vista da reutilização, suportando a extensão ou especificação do comportamento do *framework*, através do uso da herança (de interface ou de dados) e da ligação dinâmica.
- A *ligação dinâmica* (*"dynamic binding"*). Um *framework* pode, de certa forma, ser considerado uma biblioteca concebida para ligação dinâmica. Enquanto a utilização de uma biblioteca compreende apenas chamadas feitas no sentido aplicação-biblioteca, nos *frameworks* as chamadas, em geral, fluem no sentido oposto, de acordo com o *Princípio de Hollywood*: *"Don't call us, we'll call you!"*;
- Os *padrões de desenho*, que constituindo blocos de construção de *software* de nível superior às dasses abstractas e concretas, e representando soluções abstractas para problemas comuns e recorrentes de desenho, serão referências importantes no desenvolvimento de *frameworks*.

- Os *contratos*, que especificam colaborações entre objectos, de modo a permitir derivar um desenho concreto a partir de um desenho abstracto.

4.3.2 Caracterização de *frameworks*

Um *framework OO* pode ser caracterizado em diversas dimensões [Taligent 1994]. As mais relevantes são o *domínio problema*, a *estrutura interna* e a *utilização pretendida*.

Relativamente ao *domínio problema*, os *frameworks* podem ser divididos em: *aplicacionais*, quando cobrem funcionalidade que pode ser aplicada em diferentes domínios; *de domínio*, quando incorporam conhecimento e perícia num domínio problema particular; *de suporte*, quando disponibilizam serviços de sistema de baixo nível.

No campo da *estruturação interna* existem diversas abordagens [Buschmann, Meunier 1994], desde, por exemplo, estruturas de camadas, para domínios decomponíveis em grupos de tarefas com níveis distintos de abstracção, a estruturas do tipo *"broker"*, para sistemas distribuídos, com componentes desacoplados que interagem através de chamadas remotas cliente-servidor.

Um *framework* pode ser *usado* de duas maneiras:

- Derivando novas classes, numa abordagem geralmente designada de *caixa branca*. A adaptação do *framework* é realizada através de derivação de classes e re-escrita de operações.
- Instanciando e combinando classes existentes, numa abordagem designada de *caixa preta*. A adaptação do *framework* é realizada através de composição de objectos. A forma como os objectos podem ser combinados é descrita pelo *framework*, mas o que este faz depende de que objectos o utilizador passa ao *framework*.

A segunda abordagem será, em geral, de utilização mais fácil, com maior eficiência de execução e um maior grau de desacoplamento entre componentes. No entanto, apresenta-se limitativa no aspecto da extensibilidade. Por isso, algumas abordagens propõem a construção de um *framework* com uma base do tipo caixa branca e uma camada do tipo caixa preta, por forma a permitir que este seja simultaneamente extensível e de fácil utilização.

4.3.3 Desenvolvimento de *frameworks*

Em [Wilson, Wilson 1993] e [Johnson 1993] são apresentados alguns processos para desenvolvimento de *frameworks*, que, basicamente, são constituídos por ciclos de desenvolvimento e utilização, e diferem, sobretudo, no ponto de partida para construção do *framework*: experiência anterior com aplicações ou análise do domínio.

No primeiro caso, o ciclo parte de experiência anterior de desenvolvimento de aplicações, identificando e incorporando aspectos comuns e experiência num *framework*. As aplicações são, então, adaptadas para utilizarem o *framework*. A experiência entretanto obtida é utilizada para posterior manutenção do *framework*.

Num processo baseado na análise do domínio, o trabalho base consiste na identificação e compreensão das abstracções chave do domínio. O *framework* é, então, desenvolvido com uma aplicação de teste. A evolução do *framework* é realizada em interacção com o desenvolvimento e evolução das aplicações que o utilizam.

Quando o processo envolve *padrões de desenho*, a base para o desenvolvimento é, tipicamente, uma aplicação inicial no domínio problema, à qual se aplicam sistematicamente *padrões* para criar o *framework*. Segue-se, então, o processo interactivo de desenvolvimento de aplicações e evolução do *framework*.

4.3.4 Utilização de *frameworks*

A utilização prática de *frameworks*, num número já assinalável de casos, tem permitido confirmar algumas das vantagens e desvantagens esperadas *a priori*.

Em [Mattsson 1996] são destacados alguns dos pontos fortes da sua aplicação:

- Ao nível da codificação, verifica-se um decréscimo real no volume de código a desenvolver, sempre que exista um elevado grau de semelhança entre a funcionalidade necessária para uma aplicação e a funcionalidade disponibilizada pelo *framework*. Um *framework* pode, ainda, disponibilizar, à partida, maior eficiência para o *software* do que a disponibilizada por um projecto normal de desenvolvimento.
- O facto de um *framework* ter já sido submetido a processos de teste e correcção permite uma redução do esforço nesta área. Por outro lado, a utilização de *frameworks* introduz melhorias ao nível da manutenção, uma vez que um erro corrigido no *framework* fica imediatamente corrigido em todo o *software* dele derivado.

- A utilização de *frameworks* permite a reutilização de desenho e não só de código, facilitando a transferência de conhecimento e experiência a esse nível. Permite ainda uma maior concentração dos seus utilizadores nas respectivas áreas de conhecimento especializado, uma vez que os *frameworks* tipicamente contemplarão, à partida, os aspectos mais comuns da funcionalidade do domínio aplicacional.

O autor referido destaca ainda alguns potenciais pontos fracos:

- Os processos de correcção podem ser afectados pela dificuldade em distinguir erros do *framework* de erros da aplicação. Se os erros se situarem no *framework*, a sua correcção poderá mesmo ser inviável para o utilizador.
- A "retrocompatibilidade" entre evoluções do *framework* e as aplicações dele derivadas pode ser difícil de manter.
- A generalidade e a flexibilidade do *framework* podem ser impeditivas de um nível adequado de eficiência em aplicações particulares.
- O desenvolvimento de um bom *framework* exige uma forte experiência no respectivo domínio aplicacional.

4.4 Abordagens orientadas por objectos para meta-heurísticas

4.4.1 Introdução

De uma forma talvez um pouco simplificadora, poder-se-ia afirmar que duas ordens de motivações principais têm estado na base das abordagens Orientadas por Objectos (OO) no campo das meta-heurísticas, ou de forma mais geral, dos métodos para resolução de Problemas de Optimização Combinatória:

1. na linha da análise de [Nievergelt 1994], a necessidade de intensificar a ponte entre a investigação e a aplicação em problemas reais, através da implementação de sistemas informáticos: *"No systems, no impact!"*;
2. de acordo com [Ferland et al. 1996], a necessidade de criar *software* mais genérico que permita tornar mais eficiente a comparação de algoritmos sobre um problema específico, e minimizar o esforço de programação aí envolvido.

A utilização do paradigma OO nas várias fases de desenvolvimento de *software* - análise, desenho e implementação - permite, de acordo com um crescente número de autores, responder a estes desafios.

Relativamente ao primeiro desses desafios, em [Fink et al. 1998a] são apontadas as principais razões que, de um ponto de vista prático, justificam este tipo de preocupações. O objectivo último da Investigação Operacional é o apoio à decisão em problemas reais, que se caracterizam por uma grande diversidade e volatilidade. Juntamente com a necessidade de dispor de sistemas informáticos de fácil utilização, este facto coloca fortes exigências ao desenvolvimento de *software*, que deverá ser fortemente apoiado. No entender destes autores, resulta daqui uma estratégia que passa pela construção de componentes de *software* reutilizáveis que dêem corpo ao conhecimento gerado pela investigação.

Em [Ferland et al. 1996] é apresentado um trabalho pioneiro na aplicação do paradigma OO a meta-heurísticas, enquadrado sobretudo pela segunda das motivações referidas. Desde logo se destacam, igualmente, diversos aspectos relevantes também do ponto de vista prático, em particular a modularidade e a racionalidade das estruturas de *software*, permitindo reutilização e aumento de eficiência no seu desenvolvimento.

As principais vantagens e inconvenientes gerais da aplicação de abordagens OO a esta área são sintetizadas em [Schaerf et al. 1999]: entre as vantagens, são destacadas a possibilidade de reutilização, a clareza conceptual (separação entre problemas e técnicas algorítmicas), a possibilidade de composição de técnicas, a facilidade de prototipagem e experimentação e o estabelecimento de uma metodologia para a concepção de algoritmos; as principais desvantagens apontadas são alguma perda de flexibilidade e de eficiência computacional.

Nos últimos anos têm surgido várias abordagens na área da optimização, com fortes preocupações de reutilização de *software*. Em [Fink et al. 1998a] são apontadas algumas referências relativas a essas abordagens.

No contexto particular da Optimização Combinatória, um esforço pioneiro deu origem ao *framework* ABACUS [Thienel 1995], desenvolvido dentro do paradigma OO e com implementação na linguagem C++. Este *framework* tem por objecto problemas formulados como Programação Inteira Mista, para cuja resolução disponibiliza algoritmos de *branch-and-bound* baseados em relaxação linear, que podem ser complementados com geração dinâmica de planos de corte ou de colunas. O *framework* promove a (re)utilização destas técnicas, permitindo ao utilizador concentrar-se apenas nas partes específicas ao problema, ou seja, na geração de planos de corte ou de colunas e nas heurísticas primais.

Um sistema com assinalável impacto comercial é o ILOG Solver [Puget 1994, 1995, ILOG 1997], também desenvolvido em C++, que disponibiliza componentes para implementação de abordagens baseadas em Programação Lógica por Restrições, bem como Programação Matemática, desde a fusão, em 1997, com o CPLEX, uma das mais conhecidas bibliotecas de *software* nesta área.

Na área mais específica das meta-heurísticas, o desenvolvimento de *frameworks* tem-se estruturado em torno de uma questão central: a separação entre as características específicas dos problemas e os algoritmos meta-heurísticos [Fink et al. 1998a, Fink et al. 1998b]. Com efeito, aspectos como o espaço de soluções ou a estrutura de vizinhança dependem fortemente do problema em consideração e apresentam uma grande diversidade de especificações possíveis. Uma apreciação das abordagens encontradas na literatura mostra que esta separação tem tipicamente sido realizada de uma das seguintes formas:

1. Os componentes meta-heurísticos genéricos são implementados com classes abstractas correspondentes aos componentes dependentes do problema. Para a aplicação a um problema particular, são construídas as classes concretas relativas ao problema, derivadas das referidas classes abstractas. O polimorfismo dinâmico permite aos componentes genéricos lidar com as classes derivadas.
2. Os componentes meta-heurísticos genéricos são implementados com mecanismos de polimorfismo estático, em particular, o mecanismo *"template"* em C++ (ver, a este respeito, a descrição do *framework HOTFRAME* [Fink et al. 1998b] em 4.4.5). Para a aplicação a um problema particular, são construídas as classes concretas relativas ao problema, que irão parametrizar os componentes genéricos.

As abordagens encontradas na literatura serão descritas com algum pormenor nas subsecções seguintes. Na Tabela 4.1 apresenta-se uma caracterização sumária dessas abordagens relativamente aos seguintes aspectos: princípio utilizado na separação algoritmo/problema, organização geral do *framework*, algoritmos implementados, problemas abordáveis utilizando o *framework* e linguagem de programação utilizada na respectiva implementação.

Tabela 4.1 Abordagens OO para meta-heurísticas

Nome	Referências	Princípio	Organização	Algoritmos	Problemas	Linguagem
NST-ATP	[Ferland et al. 1996]	Herança	Duas hierarquias de classes: problemas e algoritmos.	*Iterative Improvement*, *Exchange Procedure* [Ferland, Lavoie 1992], TS e SA	Tipo afectação (*"Assignment-type"*)	*Object-oriented Turbo Pascal*
Class Library for Heuristic Search Optimization	[Woodruff 1997]	Herança	Classes base: problema, algoritmo, solução e vizinhança.	GA, *Descent*, TS, SA e estratégias híbridas	Genérico	*C++*
Searcher	[Andreatta et al. 1998]	Herança	Arquitectura baseada em *padrões de desenho*. Classes base: problema, algoritmo construtivo, pesquisa heurística, solução, modelo de incremento e modelo de movimento.	GRASP, VNS e TS	Genérico	*C++*
HOTFRAME	[Fink et al. 1998b]	Polimorfismo estático	Componentes relacionados com o problema: problema, solução e vizinhança. Componentes com algoritmos.	*Steepest Descent*, SA e TS	Genérico	*C++*
Local ++	[Schaerf et al. 1999]	Herança, polimorfismo estático	Algoritmos: componentes genéricos, organizados em hierarquia, através de herança. Aplicação a problemas específicos através de classes concretas para parametrização dos algoritmos: soluções e movimentos.	*Hill Climbing*, SA e TS Algoritmos compostos: *Tandem Search* e *Hybrid Search*	Genérico	*C++*
TabOOBuilder	[Graccho, Porto 1999]	Herança	Classes relacionadas com problemas do tipo afectação: *resource*, *item*, *evaluator*. Classes relacionadas com TS.	TS	Tipo afectação	*Java*

Do ponto de vista da reutilização de *software*, a análise desta tabela permite confirmar o interesse deste tipo de abordagens: por um lado, a maioria das abordagens efectivamente alcança flexibilidade relativamente aos tipos de problemas a abordar; por outro, em diversos casos, os próprios algoritmos beneficiam de reutilização, através da sua organização, por exemplo, em hierarquias de algoritmos.

Em relação aos algoritmos cobertos por esta análise, saliente-se que existe apenas um *framework* com GA e dois com abordagens híbridas, o que sugere a existência de direcções de trabalho ainda em aberto e com forte potencial. Na área das meta-heurísticas multiobjectivo, não há referência a qualquer iniciativa.

Um facto igualmente relevante, do ponto de vista da implementação, é a quase unânime selecção de C++ como linguagem de programação, o que não será certamente indissociável da crescente generalização da utilização desta linguagem, que cobre uma larga maioria dos conceitos de OO, e para a qual existem compiladores eficientes em praticamente todas as plataformas de *hardware* e sistemas operativos [Thienel 1995].

4.4.2 *NST-ATP*

A primeira abordagem OO para implementação de meta-heurísticas encontrada na literatura foi proposta em [Ferland et al. 1996], com a designação NST-ATP (*"Neighbourhood Search Techniques - Assignment-Type Problems"*). Os autores referidos apresentam um *framework* com diversos algoritmos de optimização baseados em pesquisa local, para problemas do tipo afectação ("*assignment-type*"). O problema genérico pode ser definido da seguinte forma: considerando um conjunto de itens e um conjunto de recursos, afectar os itens aos recursos por forma a optimizar uma função objectivo e satisfazer um conjunto de restrições.

O *framework* estrutura-se em duas hierarquias de classes: uma primeira, que estabelece um interface com as estruturas de dados para o problema, disponibilizadas pelo utilizador, e uma segunda, que inclui quatro algoritmos de optimização baseados em pesquisa local: *Iterative Improvement*, *Exchange Procedure* [Ferland, Lavoie 1992], TS e SA.

A primeira hierarquia permite obter a informação relevante a partir das estruturas de dados do utilizador, e transformá-la no formato específico exigido pela segunda hierarquia. Por outro lado, poderá também ser aplicada numa eventual transformação de sentido inverso. A principal classe da hierarquia possui algum nível de abstracção, o que permite a sua aplicação mais específica a cada problema particular, através da utilização de polimorfismo.

A segunda hierarquia contém quatro classes distintas, correspondentes aos quatro algoritmos disponibilizados. Estas classes são derivadas de uma classe base, usada apenas para estabelecer parâmetros comuns às diferentes técnicas, e controlar o tempo de execução e o critério de paragem. As técnicas podem ser usadas indiferentemente, uma vez que estão adaptadas a qualquer problema para o qual tenha sido especificado o interface referido na primeira hierarquia. Estas classes possuem igualmente algum nível de abstracção, permitindo a um utilizador criar, através da utilização de polimorfismo, uma qualquer variante que pretenda implementar.

A consideração de uma estrutura de problema específica permite evitar a necessidade de definição de soluções, vizinhanças e movimentos, por parte do utilizador do *framework*.

Esta abordagem geral foi a primeira a permitir comparar a eficiência relativa de diversos algoritmos na resolução de um problema específico, bem como a sua adaptação através de variantes ou alterações dos parâmetros, por intermédio de intervenções com um esforço de programação mínimo. O *software* foi desenvolvido em *Object-Oriented Turbo Pascal* para plataformas do tipo PC.

4.4.3 *"A Class Library for Heuristic Search Optimization"*

Em [Woodruff 1997] é apresentada uma biblioteca de classes em C++ para meta-heurísticas puras e híbridas. Apesar da designação de biblioteca, o *software* exibe integralmente as características de um *framework*. De um ponto de vista teórico, a organização da biblioteca pode ser vista como correspondendo a uma taxonomia de meta-heurísticas, lançando, assim, alguma luz sobre as relações entre estas e as formas como podem ser combinadas. De um ponto de vista prático, os objectivos da biblioteca são a avaliação e a aplicação prática de meta-heurísticas.

Um conjunto de classes base, puramente abstractas, estrutura a biblioteca (Figura 4.1):

- *HS_Base*: base para algoritmos, da qual são derivadas novas classes base:
  - *GA_Base* correspondente aos GA, da qual por suas vez são derivadas duas classes concretas correspondentes a duas abordagens distintas: geracional (*Elite_GA*) e estacionária (*Steady_GA*).
  - *NR_HS_Base*, correspondente a abordagens baseadas em pesquisa local, da qual são derivadas duas classes concretas com estratégias elementares de pesquisa local (*Descent* e *RR_Descent*) e duas classes abstractas, que servirão de base para algoritmos de SA (*SA_Base*) e TS (*Tabu_Base*).

- *Problem_Base*: base para a definição de problemas.
- *Neighborhood_Base*: base para a definição de vizinhanças, derivada de *Mixed_Solution*, classe que representa as soluções e que será o elemento constituinte da população (classe *Population*) nos GA. As soluções são obrigatoriamente representadas por vectores de inteiros.

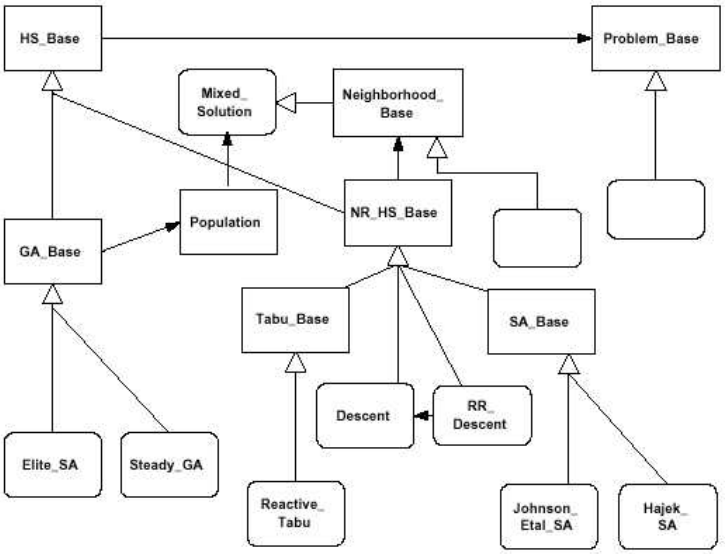


Figura 4.1 Estrutura de classes de *"A Class Library for Heuristic Search Optimization"* [Woodruff 1997]

A biblioteca assenta, portanto, numa estrutura hierárquica, permitindo a realização de extensões ou combinações de classes para produzir novas estratégias de pesquisa, através da utilização do interface das classes abstractas.

Para ter imediatamente à disposição um conjunto de algoritmos, um utilizador da biblioteca apenas necessita de disponibilizar uma definição de um problema e uma estrutura de vizinhança. Por exemplo, a utilização da biblioteca para resolver um problema com uma meta-heurística baseada em pesquisa local envolve a definição de uma classe *problema*, que deverá ter *Problem_Base* como classe base pública, e de uma classe *vizinhança*, que, de forma idêntica, deverá ter *Neighborhood_Base* como classe base pública. A utilização de herança permitirá, então, que classes de algoritmos, implementadas com uma classe base de vizinhança abstracta, possam manipular, sem conhecer os respectivos detalhes, classes de vizinhança derivadas dessa classe base.

4.4.4 *Searcher*

Em [Andreatta et al. 1998] é proposto o *framework Searcher*, desenvolvido em C++, para meta-heurísticas baseadas em pesquisa local. O objectivo do *framework* é disponibilizar uma arquitectura base para a implementação e comparação de diferentes estratégias. A sua

arquitectura apresenta como principal particularidade o facto de assentar na aplicação de *padrões de desenho*.

O *framework* é particularmente apropriado para utilização em situações que envolvam: diversos métodos para construção da solução inicial, estruturas de vizinhança, ou critérios de selecção de movimentos; algoritmos construtivos utilizando heurísticas de pesquisa local subordinadas, para melhoria de soluções parcialmente construídas; heurísticas de pesquisa local com modelos de vizinhança dinâmicos.

Os autores referidos utilizam para a descrição do *framework* uma adaptação do *template* de descrição de *padrões de desenho* de [Gamma et al. 1995]. Para efeitos de apresentação geral do *framework*, destacam-se dessa descrição a estrutura de classes e as suas colaborações.

As classes base do *framework*, cuja estrutura é apresentada na Figura 4.2, são as seguintes:

- *Cliente* (*Client*): contém a instância do problema a resolver, os métodos de pré-processamento e pós-processamento a aplicar e os dados para criar a Estratégia de Pesquisa (*SearchStrategy*) a usar.
- *Solução* (*Solution*): encapsula a representação de uma solução para o problema e define o interface a utilizar pelos algoritmos para construir e modificar uma solução.
- *Estratégia de Pesquisa* (*SearchStrategy*): constrói e lança os algoritmos Estratégia Construtiva (*BuildStrategy*) e Pesquisa Local (*LocalSearch*).
- *Estratégia Construtiva* (*BuildStrategy*): encapsula algoritmos construtivos em subclasses concretas.
- *Pesquisa Local* (*LocalSearch*): encapsula algoritmos de pesquisa local em subclasses concretas.
- *Incremento* (*Increment*): agrupa a informação necessária à modificação da representação interna de uma Solução por algoritmos construtivos.
- *Movimento* (*Movement*): agrupa a informação necessária à modificação da representação interna de uma Solução por algoritmos de pesquisa local.
- *Modelo de Incremento* (*IncrementModel*): modifica uma Solução de acordo com um pedido de uma Estratégia Construtiva.
- *Modelo de Movimento* (*MovementModel*): modifica uma Solução de acordo com um pedido de uma Pesquisa Local.

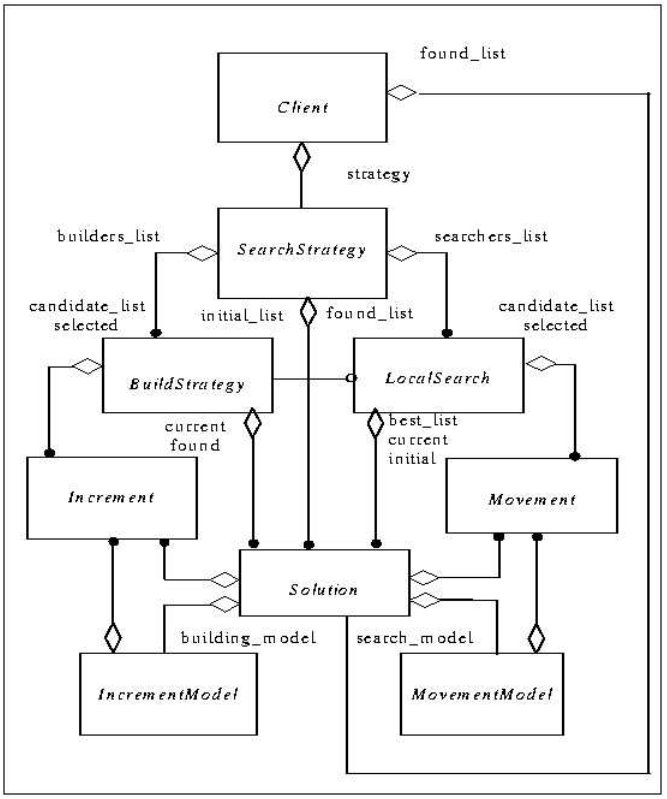


Figura 4.2 Diagrama de classes do *framework Searcher* [Andreatta et al. 1998]

As colaborações mais importantes entre estas classes, representadas no diagrama de sequência da Figura 4.3, são as seguintes:

- Um Cliente recorre a uma Estratégia de Pesquisa para obter uma Solução para uma instância de um problema.
- A Estratégia de Pesquisa é composta por, pelo menos, uma Estratégia Construtiva, que produz uma solução inicial, e uma Pesquisa Local, que a procura melhorar.
- A construção de uma solução é delegada pela Solução no Modelo de Incremento, enquanto as modificações baseadas em vizinhanças são delegadas no Modelo de Movimento. Estas classes obterão, respectivamente, Incrementos e Movimentos para modificação da Solução.

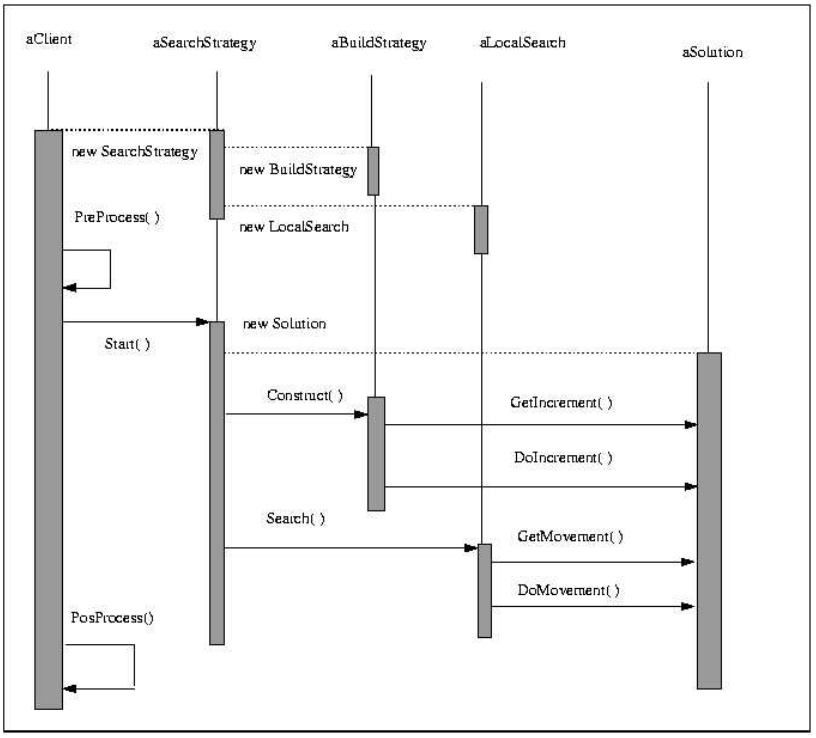


Figura 4.3 Diagrama de sequência para o *framework Searcher* [Andreatta et al. 1998]

A arquitectura do *framework* assenta na aplicação de três *padrões de desenho* de [Gamma et al. 1995]: *Estratégia* (*"Strategy"*), *Método Template* (*"Template Method"*) e *Estado* (*"State"*). Para completar a apresentação geral aqui realizada, resumem-se de seguida os principais aspectos envolvidos na aplicação destes padrões de desenho:

- A aplicação do *padrão Estratégia* resulta em isolar o problema e a estratégia de pesquisa em classes distintas, encapsulando assim a informação de pesquisa e permitindo maior facilidade de construção e extensão de uma família de estratégias de pesquisa. Resulta ainda na separação entre as classes Estratégia de Pesquisa e as classes Estratégia Construtiva e Pesquisa Local. Deste modo, famílias de algoritmos para a construção de soluções iniciais, e famílias de algoritmos para pesquisa local podem ser desenvolvidas com facilidade e de uma forma independente.

- O *padrão Método Template* é usado na derivação de classes concretas de algoritmos a partir da classe base Pesquisa Local. Este *padrão* comportamental define o esqueleto de um algoritmo como uma operação associada a uma classe abstracta, deixando as subclasses refinar certos passos do algoritmo, em particular os passos correspondentes ao comportamento que pode variar. Tal possibilita, no *framework*, a realização de especializações de um algoritmo genérico de pesquisa local, correspondentes a algoritmos de *Iterative Improvement* ou TS.

- O *padrão* Estado é usado na implementação de vizinhanças dinâmicas. Na sua aplicação, a classe Solução delega no Modelo de Movimento a construção de diferentes tipos de vizinhanças. A implementação de vizinhanças dinâmicas é, então, realizada a partir do uso de polimorfismo sobre o Modelo de Movimento, dentro da sua hierarquia de classes. Esta delegação de algoritmos poderia novamente ser considerada como uma aplicação do *padrão* Estratégia. Contudo, como é a própria classe responsável pela construção da vizinhança que se altera durante o processo, a delegação identifica-se melhor com uma aplicação do *padrão* Estado.

Relativamente às possibilidades de extensão do *framework*, os autores destacam a extensão aos GA, para a qual sugerem a criação da classe População, com o formato de conjuntos de soluções e contendo as operações binárias que definem novas soluções. As relações existentes no *framework* manter-se-iam; novas relações entre Pesquisa Local, População e Solução deveriam ser incluídas.

4.4.5 *HOTFRAME ("Heuristic OpTimization FRAMEwork")*

O *framework HOTFRAME*, apresentado em [Fink et al. 1998b], foi implementado em C++, e tem como principais utilizações alvo a aplicação prática e a investigação na área da optimização com meta-heurísticas.

A principal particularidade deste *framework* é a utilização de polimorfismo estático, disponibilizado em C++ através do mecanismo *"template"*. Neste mecanismo, um parâmetro tipo é incluído na forma *"<...>"* numa classe *"template"*. Por exemplo, *X_P<T, Range>* representará instâncias para um problema específico *X_P*, com o tipo *T* definindo os parâmetros do problema (por exemplo, valores inteiros ou reais) e o tipo *Range* definindo o máximo número de objectos.

Na parte do *framework* que contempla os componentes específicos aos problemas, são considerados os seguintes tipos abstractos:

- uma classe problema X_P;
- uma ou mais classes solução X_S;
- para cada classe solução X_S, uma ou mais classes de informação da solução X_S_I;
- para cada classe solução X_S, uma ou mais classes de vizinhança X_S_N;

- para cada classe de vizinhança X_S_N, uma ou mais classes X_S_A com os atributos de movimento correspondentes;

Estes tipos são visualizados pelo diagrama de classes simplificado apresentado na Figura 4.4.

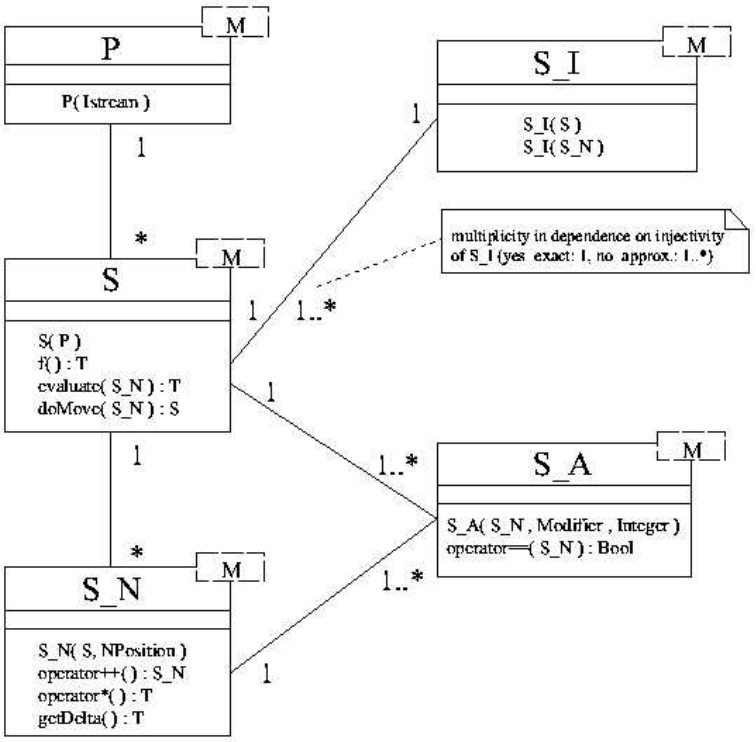


Figura 4.4 Diagrama de classes para o *framework HOTFRAME* [Fink et al. 1998b]

Relativamente à parte algorítmica do *framework*, os componentes são implementados através de classes *"template"*, parametrizadas com os componentes específicos ao problema anteriormente descritos. Por exemplo, um algoritmo de *"Steepest Descent"* deverá ser parametrizado com um tipo de solução e um tipo de vizinhança. A instanciação de um componente deste tipo para um problema com um tipo de solução *TSolução* e um tipo de vizinhança *TVizinhança* seria realizada da seguinte forma:

SteepestDescent< TSolução < T, Range >, TVizinhança < T, Range > >.

O *framework* baseia-se, portanto, em primeiro lugar, na generalização através de *"templates"*. Os mecanismos de herança são utilizados apenas de forma secundária. Este tipo de abordagem apresenta, em relação a abordagens baseadas em herança, vantagens potenciais ao nível da eficiência de execução, do desacoplamento dos componentes do *framework*, e do compromisso entre flexibilidade e facilidade de reutilização do tipo *caixa-preta*. A herança é essencialmente usada para destacar comportamento comum para problemas semelhantes.

4.4.6 *Local++*

Em [Schaerf et al. 1999] é apresentado um *framework*, designado *Local ++*, que se destina a ser usado como ferramenta genérica para o desenvolvimento e implementação de algoritmos de pesquisa local em C++. Uma particularidade importante no *framework* é o facto de disponibilizar, a um nível abstracto, a definição de *solvers* compostos.

O *framework* estrutura-se em torno da utilização do *padrão de desenho Método Template* [Gamma et al. 1995], já referido anteriormente, permitindo especificar e implementar as partes invariantes de vários algoritmos de pesquisa local. O núcleo do *framework* é, consequentemente, constituído por uma hierarquia de *solvers*, apresentada na Figura 4.5 As classes mais baixas da hierarquia representam os algoritmos em si, e será a partir delas que se derivarão classes concretas.

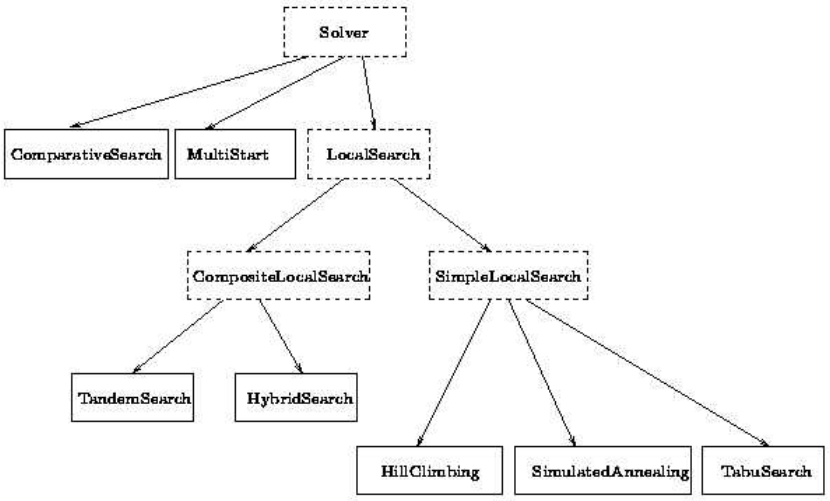


Figura 4.5 Estrutura geral do *framework Local++* [Schaerf et al. 1999]

A classe de topo *Solver* encapsula as características comuns a qualquer *solver*. *LocalSearch* constitui a base para as técnicas de pesquisa local, que são classificadas em técnicas simples (*SimpleLocalSearch*), que fazem uso de um único tipo de movimento e uma única estratégia, e compostas (*CompositeLocalSearch*), correspondendo aos casos restantes. O *Local++* compreende três técnicas simples de pesquisa local, *Iterative Improvement* (*HillClimbing*), SA (*SimulatedAnnealing*) e TS (*TabuSearch*), e duas compostas, Pesquisa em Sequência (*TandemSearch*) e Pesquisa Híbrida (*HybridSearch*).

O *framework* inclui ainda duas classes baseadas em múltiplas execuções de algoritmos subjacentes. *MultiStart* executa um *solver* básico, um determinado número de vezes, apresentando como resultado a melhor das soluções encontradas nas várias execuções. *ComparativeSearch* utiliza um vector de *solvers* simples, executados de forma independente, e apresenta como resultado global o melhor de todos os resultados individuais.

O interface entre o problema a tratar e os algoritmos disponibilizados pelo *framework* é realizado de duas formas:

- considerando tipos abstractos em componentes algorítmicos genéricos, através de classes *"template"* para as soluções e para os movimentos;
- implementando partes do interface de classes abstractas, nos casos da construção da solução inicial, do cálculo do valor da função objectivo para uma solução, do procedimento de realização de movimentos e do cálculo da variação no valor da função objectivo correspondente a um movimento.

A parte algorítmica corresponde à hierarquia de *solvers* já apresentada, na qual a utilização do *padrão de desenho Método Template* se realiza a diversos níveis. A aplicação com maior impacto situa-se ao nível da Pesquisa Local Simples, com a definição de uma função *Run()*, que exprime o mecanismo geral da pesquisa local (Algoritmo 4.1).

```
template<class Input, class Output, class State, class Move>
void SimpleLocalSearch<Input,Output,State,Move>::Run()
{ InitializeRun();
  while (ContinueSearching())
    { UpdateIterationCounter();
      SelectMove();
      if (AcceptableMove())
        { MakeMove();
          StoreMove();
          if (LowerBoundReached())
                break;
        }
    }
  TerminateRun();
}
```

Algoritmo 4.1 Mecanismo geral da pesquisa local no *framework Local++*

Este mecanismo geral não necessitará de ser refinado nas subclasses, que diferirão entre si apenas na forma como definem as partes do interface invocadas, mas não implementadas, a este nível:

- *InitializeRun()* e *TerminateRun()* consistem das acções a tomar, respectivamente, antes e depois do ciclo de pesquisa principal.
- *ContinueSearching()* será verdadeira ou falsa de acordo com o critério de paragem do algoritmo específico. *LowerBoundReached()* é usada para terminar a pesquisa no caso de ter sido atingida uma solução admissível como solução final.

- *SelectMove()* e *AcceptableMove()* executam, em conjunto, a estratégia de selecção de movimentos. A primeira selecciona um movimento, enquanto a segunda verifica se o movimento seleccionado é aceitável.
- *StoreMove()* efectua operações necessárias após a execução de um movimento, relacionadas com o algoritmo específico. Tal poderá consistir, por exemplo, no armazenamento do movimento numa estrutura de dados adequada ou, simplesmente, em assinalar que um movimento foi efectuado.

Em [Schaerf et al. 1999] ilustra-se esta aplicação para uma versão de *Hill Climbing* e para TS. Para complementar a apresentação desta abordagem, sintetizam-se de seguida os aspectos mais importantes do caso particular de *Hill Climbing*. Na implementação realizada, em cada iteração é escolhido aleatoriamente um movimento, que será aceite se não degradar o valor da função objectivo. O critério de paragem escolhido consiste em interromper a execução ao fim de um determinado número de iterações sem melhoria do valor da função objectivo. Estas especificações correspondem à seguinte implementação das funções atrás referidas:

- *InitializeRun()* inicializa variáveis de contagem de iterações.
- *ContinueSearching()* verifica se decorreu o número pré-definido de iterações sem melhoria da função objectivo.
- *SelectMove()* gera aleatoriamente um movimento.
- *AcceptableMove()* verifica se a execução do movimento não degrada o valor actual da função objectivo.
- *StoreMove()* actualiza o valor da função objectivo e armazena o número da iteração actual, no caso de o movimento ser aceite.

Na parte relativa aos algoritmos compostos, os autores descrevem apenas a Pesquisa Sequencial, na qual dois algoritmos simples distintos, com duas estruturas de vizinhança distintas, são alternados. Este tipo particular de algoritmo exige um critério de paragem externo aos algoritmos simples, que terão os seus critérios de paragem próprios. A disponibilização das duas estruturas de vizinhanças, uma para cada um dos algoritmos simples, é realizada através de duas classes *template* distintas.

4.4.7 *TabOOBuilder*

Um *framework* desenvolvido em Java, designado *TabOOBuilder*, é apresentado em [Graccho, Porto 1999]. O *framework* tem por objectivo a construção de aplicações baseadas em TS, para problemas do tipo afectação, à semelhança de [Ferland et al 1996].

A cada um dos conceitos da estrutura do problema foi associada uma classe específica no *framework*. Uma classe *Problem* funciona como contentor para as classes *Resource* e *Item*, responsáveis por disponibilizar os atributos necessários às suas próprias descrições, bem como das eventuais restrições associadas. A avaliação da função objectivo é colocada num interface aparte, denominado *Evaluator*, que especifica unicamente o protocolo de acesso ao cálculo do custo. Este cálculo deverá ser implementado pelo utilizador do *framework*. A consideração de uma estrutura de problema particular permite ter classes correspondentes às soluções do problema, às vizinhanças e aos movimentos, que são inteiramente independentes das características particulares de um problema específico. Essas classes são completamente disponibilizadas pelo *framework*, não necessitando de esforço de implementação por parte do utilizador.

A parte algorítmica é composta por TS, permitindo definir diferentes configurações e estratégias para este algoritmo. Um aspecto chave na TS é o uso de informação da história da pesquisa. As estruturas de memória que registam a evolução da pesquisa podem ser classificadas de acordo com as características efectivamente registadas: proximidade, frequência, qualidade e influência. A memória de proximidade está relacionada com o conceito de lista tabu, para a qual o *framework* reserva a classe correspondente. Para as restantes estruturas de memória, com funcionalidades distintas, o *framework* disponibiliza igualmente as classes correspondentes.

Um outro aspecto chave é a utilização de estratégias de diversificação e intensificação, para as quais o *framework* disponibiliza os interfaces correspondentes. O *framework* disponibiliza ainda um interface para estratégias gerais, que permitem determinar quando e como diversificar e intensificar, e estabelecer regras de paragem. As estratégias são modeladas com total independência do problema em consideração, permitindo aos utilizadores experimentar diferentes estratégias para o mesmo problema, ou mesmo uma mesma estratégia para problemas distintos.

4.5 Outras abordagens para meta-heurísticas

Na literatura destacam-se ainda dois outros trabalhos relevantes no âmbito desta dissertação. Dada a importância que a utilização do *padrão de desenho Método Template* [Gamma et al. 1995] assume em duas das abordagens OO, terá interesse apresentar o *template* proposto

em [Vaessens et al. 1995], com o objectivo de captar a maioria das variantes de algoritmos de pesquisa local propostos na literatura. Uma iniciativa de carácter distinto, mas igualmente relevante do ponto de vista do apoio à concepção e implementação de meta-heurísticas, é a linguagem *Localizer* [Michel, van Hentenryck 1998], que permite modelar algoritmos de pesquisa local numa notação próxima das descrições informais deste tipo de procedimentos, apresentadas em publicações científicas.

4.5.1 *Template* de pesquisa local

O *template* apresentado em [Vaessens et al. 1995] foi concebido por forma a captar a maioria das variantes de algoritmos de pesquisa local propostas na literatura, e, assim, permitir classificar os vários algoritmos existentes, identificar novas abordagens e sugerir variantes inovadoras.

A ideia base do trabalho referido é partir do algoritmo de pesquisa local mais básico, o *Iterative Improvement*, que termina a sua execução no primeiro óptimo local encontrado, e perspectivar a maioria dos restantes algoritmos de pesquisa local como resultantes de várias alterações a esse algoritmo, visando ultrapassar as suas limitações. O *template* baseia-se, portanto, no *Iterative Improvement*, que é generalizado das seguintes formas:

- A pesquisa pode realizar-se em diversos níveis, com especificações próprias.
- A solução actual única é substituída por uma população de soluções actuais.
- A estrutura de vizinhança associada a uma solução única é substituída por uma estrutura de vizinhança associada a um agrupamento (*"cluster"*) de soluções.

Em cada nível de pesquisa l a população terá uma dimensão p_l, e os agrupamentos de soluções uma dimensão c_l. Uma população no nível l é um tuplo de dimensão p_l, $P \in S^{p_l}$, de soluções, que representa o estado actual da pesquisa no nível l. A pesquisa diz-se *pontual* se $p_1 = 1$ e *populacional* nas restantes situações. Um agrupamento no nível l é um tuplo de dimensão c_l, $C \in S^{c_l}$, de soluções, tal que a cada agrupamento é associada uma vizinhança. Ou seja, existe uma função de *hiper-vizinhança* $N_l : S^{c_l} \to 2^S$ que, para cada agrupamento C, define um conjunto $N_l(C)$ de soluções vizinhas. Se $c_l = 1$, esta função reduz-se à vizinhança normal.

A pesquisa local procede em dois passos: em primeiro lugar, é chamado o procedimento INICIALIZAR, que gera uma população inicial $P \in S^{p_l}$, e depende, em geral, do tipo de problema em consideração; de seguida, é chamado o procedimento recursivo PESQUISA LOCAL com nível l e população P como parâmetros.

Este procedimento (Algoritmo 4.2), apresenta a seguinte estrutura:

- Ao nível l, o procedimento toma como entrada uma população $P \in S^{p_l}$ e usa a função de hiper-vizinhança N_l para produzir uma nova população $P \in S^{p_l}$ como resultado. Tal é efectuado num ciclo exterior de *gerações* e um ciclo interior de *iterações*.
- O ciclo de *gerações* cria sucessivas populações, até que seja satisfeito um critério de paragem. Em cada geração, o procedimento GERAR AGRUPAMENTOS constrói, a partir da população actual P, um conjunto finito C de agrupamentos $C \in P^{c_l}$, ou seja, cada um dos c_l componentes $C_1, ..., C_{cl}$ de C é uma solução de P.
- Para cada agrupamento $C \in \mathsf{C}$, é realizado um ciclo de *iterações* até que seja satisfeito o respectivo critério de paragem. Cada iteração começa com uma chamada ao procedimento GERAR VIZINHOS, que selecciona um conjunto finito $Q \in (N_l(C))^{p_{l+1}}$. Assim, cada um dos p_{l+1} componentes de Q é um hiper-vizinho de C; note-se que $Q \in S^{p_{l+1}}$. O procedimento PESQUISA LOCAL é então chamado recursivamente, com nível $l+1$ e população Q como parâmetros. O resultado é uma população modificada $Q \in S^{p_{l+1}}$. De seguida, o procedimento REDUZIR VIZINHOS reduz a reunião do agrupamento original C com a nova população Q a um novo agrupamento $C \in S^{c_l}$, que serve então como entrada para a iteração seguinte.
- Uma vez terminado, em cada nível l, para todo o $C \in \mathsf{C}$, o ciclo de iterações, o procedimento CRIAR agrupa as soluções encontradas na iteração final para cada $C \in \mathsf{C}$ num único conjunto $\hat{P} \subset S$. O procedimento REDUZIR POPULAÇÃO finalmente funde P e $\hat{P}$ numa nova população actual $P \in S^{p_l}$.

Para tornar finito o procedimento recursivo é necessário definir um nível base l^*. Neste nível l^*, a função CONTINUAR GERAÇÃO DE POPULAÇÕES assume o valor FALSO. Os níveis $l < l^*$ são chamados *níveis activos.*

Algoritmo PESQUISA LOCAL (l: natural; $P \in S^{p_l}$)

Enquanto CONTINUAR GERAÇÃO DE POPULAÇÕES (l)

GERAR AGRUPAMENTOS (l, P, C);

Para todo o $C \in \mathsf{C}$

Enquanto CONTINUAR ITERAÇÃO (l)

GERAR VIZINHOS (l, C, N_l, Q);

PESQUISA LOCAL ($l+1, Q$);

REDUZIR VIZINHOS (l, C, Q);

CRIAR ($l, \hat{P}$);

REDUZIR POPULAÇÃO ($l, P, \hat{P}$);

Algoritmo 4.2 *Template* de Pesquisa Local

Os autores enquadram alguns algoritmos propostos na literatura no *template* proposto: dentro das abordagens de nível único, são enquadradas na pesquisa pontual o SA, o *Threshold Accepting*, a TS e a *Variable-depth Search*, e na pesquisa populacional os GA; nas abordagens de níveis múltiplos são referidos apenas dois casos muito específicos de abordagens híbridas, um de pesquisa pontual e outro de pesquisa populacional. A título ilustrativo, apresenta-se a forma como é realizado o enquadramento para a TS:

- GERAR VIZINHOS selecciona deterministicamente os vizinhos, relativamente a N_1, da solução actual C_1, inspeccionando-os numa ordem pré-determinada.
- REDUZIR VIZINHOS determina, de entre as soluções em Q que não se encontram na lista tabu, ou que se encontram, mas satisfazem um critério de aspiração, aquela que tem um valor da função objectivo mínimo.
- CONTINUAR ITERAÇÃO será verdadeira enquanto se não verificar um critério de paragem, que poderá estar relacionado, por exemplo, com um limiar de qualidade para a melhor solução encontrada, ou com um número pré-determinado de iterações.

Aspectos inovadores como o adiamento da decisão da escolha do vizinho mais promissor ou a utilização de agrupamentos de dimensão superior a dois são também enquadrados pelo *template*.

4.5.2 *Localizer*

Em [Michel, van Hentenryck 1998] é proposta uma linguagem, designada *Localizer*, para modelação, a alto nível, de algoritmos de pesquisa local. Pretende-se com esta linguagem reduzir substancialmente os tempos de desenvolvimento e, simultaneamente, preservar o máximo possível da eficiência de implementações dedicadas. A linguagem está organizada em torno dos conceitos básicos dos algoritmos de pesquisa local e suporta os fundamentos de diversos algoritmos nela baseados, como o *Iterative Improvement*, o SA ou a TS.

O modelo computacional utilizado (Algoritmo 4.3) pode ser visto como um *template* de pesquisa local. Os diversos aspectos da linguagem foram concebidos para permitir instanciar completamente esse *template*.

```
Algoritmo Modelo Computacional da Linguagem Localizer
    Inicializar f*;
    s = Estado Inicial ();
    Para i=1 até Máximo de Tentativas
        Enquanto se cumprir um Critério Global
            Para j=1 até Máximo de Iterações
                Enquanto se cumprir um Critério Local
                    Se Admissível(s)
                        Se f(s) < f*
                            f* = f(s);
                            s* = s;
                    Seleccionar n em Vizinhança(s);
                    Se Aceitável(n)
                        s = n;
            s = Estado de Rearranque(s);
```

Algoritmo 4.3 Modelo computacional da linguagem *Localizer*

A linguagem estrutura-se em várias secções. Na Tabela 4.2 apresenta-se um primeiro conjunto dessas secções, para a definição de aspectos genéricos do problema, ou seja, aspectos não específicos à pesquisa local.

Secção	Finalidade
Data Type	Criar novas estruturas de dados.
Data	Definir as estruturas de dados que irão receber os dados da instância de problema.
Variable	Definir variáveis.
Operator	Definir operadores.
Init	Atribuir os dados da instância de problema às correspondentes estruturas de dados.
Invariant	Definir *invariantes*.
Satisfiable	Especificar critérios de admissibilidade de uma solução.
Objective Function	Definir a função objectivo.

Tabela 4.2 Secções da linguagem *Localizer* para aspectos genéricos do problema

Relativamente a estas secções, o principal aspecto a destacar é a inovação mais significativa introduzida com a linguagem: o conceito de *invariante*. Um invariante é uma expressão da forma $v = exp$, relativamente à qual a linguagem garante que, em qualquer altura da execução, o valor da variável v é o valor da expressão exp.

A título de exemplo, apresentam-se alguns casos típicos de invariantes:

- $v = \text{sum}(i \text{ in } S)a[i]$, ou seja, v é a soma dos $a[i](i \in S)$;
- $C = \{i \text{ in } S \mid a[i] = 0\}$, ou seja, C é o conjunto de $i(i \in S)$ tais que $a[i] = 0$;
- $D[i \text{ in } I] = \{j \text{ in } S \mid a[j] = i\}$, ou seja, $D[i](i \in I)$ é o conjunto de todos os $j \in S$ tais que $a[j] = i$.

Nos casos apresentados, por exemplo, sempre que um valor $a[k]$ é alterado, v, C, e $D[j]$ são actualizados. Esta actualização é conseguida através da utilização de algoritmos *incrementais* eficientes. Ainda no âmbito deste primeiro conjunto de secções, os invariantes podem ter aplicações interessantes, por exemplo, na definição da função objectivo, ou na definição de critérios de admissibilidade das soluções.

Um segundo conjunto de secções, relativas a aspectos específicos à configuração dos algoritmos de pesquisa local, é apresentado na Tabela 4.3.

Secção	Finalidade
Neighborhood	Definir a vizinhança.
Start	Construção da solução inicial.
Restart	Procedimento realizado em configurações com rearranque.
Parameter	Definição de parâmetros relacionados com a execução do algoritmo.
Global Condition	Condição global (ver modelo computacional).
Local Condition	Condição local (ver modelo computacional).

Tabela 4.3 Secções da linguagem *Localizer* para configuração de algoritmos de pesquisa local

A secção mais relevante, dentro deste segundo conjunto, é a secção para especificação das vizinhanças. A sintaxe para esta especificação é a seguinte:

select [*random* | *best*]
    op(x_1, ..., x_n)
where
    x_1 in S_1;
    ...
    x_n in S_n;
accept when
    ...

A vizinhança é definida pelo operador *op*, conjuntamente com as relações de pertença especificadas em **where**. Os invariantes poderão ser, novamente, extremamente úteis na definição destas relações de pertença. Com *random* será seleccionado aleatoriamente um elemento da vizinhança, enquanto com *best* será seleccionado o melhor elemento da vizinhança.

O critério definido em **accept when** permite determinar se o novo elemento seleccionado deve ou não substituir a solução actual. Embora possa ser definido qualquer critério, a linguagem disponibiliza alguns critérios pré-definidos: *always* **(**aceitar sempre a nova solução), *improvement* (aceitar uma nova solução apenas quando melhor do que a actual) e *no Decrease* (aceitar uma nova solução quando não pior do que a actual). É com recurso a este critério, que se torna possível, por exemplo, definir uma versão simplista de TS:

select

...

where

...

s in *S*

accept when

s not in *Tabu* ;

As instruções de **select** e **accept when** podem assumir um formato composto, considerando várias alternativas, eventualmente afectadas de uma probabilidade.

4.6 Conclusões

Em termos gerais, os dois principais incentivos para o desenvolvimento de abordagens orientadas por objectos para meta-heurísticas têm sido a aproximação de teoria e aplicação, através do desenvolvimento de sistemas abertos e de estrutura simples, que incorporem resultados teóricos, e a criação de formas mais eficientes de implementação e comparação de métodos, através de um incremento da modularidade, da racionalidade e do potencial de reutilização das estruturas de *software*.

Os quatro princípios essenciais do paradigma OO, e com um impacto decisivo no contexto deste trabalho, são o encapsulamento de dados, a abstracção de dados, o polimorfismo e a herança. O objecto é a unidade de encapsulamento de dados, consistindo de um conjunto de variáveis e um conjunto de métodos usados para alterar e aceder a essas variáveis. Um objecto aceita mensagens que invocam métodos. A assinatura de um objecto é o conjunto de mensagens que ele aceita. Uma classe é uma abstracção de dados, que exprime aspectos comuns a objectos idênticos, e que toma a forma de um padrão para criar um tipo particular de objecto. A herança de classes permite a um conjunto de classes partilhar partes comuns de uma assinatura ou implementação (variáveis e métodos). O polimorfismo é essencial para a reutilização de código, permitindo que os métodos aceitem como parâmetros objectos de tipos distintos.

O *software* OO reutilizável tem sido disponibilizado sobretudo através de bibliotecas de classes e *frameworks*. Uma biblioteca de classes agrega um conjunto de classes, eventualmente estruturado através de herança, a partir do qual é possível construir uma aplicação. Um *framework* pode ser definido como um conjunto de classes que dá corpo a um desenho (*"design"*) abstracto de soluções para um conjunto de problemas relacionados. Uma diferença essencial entre estes dois conceitos reside no facto de que os *frameworks* disponibilizam comportamento por omissão, enquanto que com as bibliotecas de classes é normalmente necessário definir toda a colaboração entre componentes. Os *frameworks*

permitem reutilização de desenho, uma área em que uma outra iniciativa tem adquirido particular relevância - os *padrões de desenho* (*"design patterns"*), descrições de objectos e classes comunicantes, que são configurados para resolver um problema de desenho genérico num contexto particular.

Várias abordagens orientadas por objectos para meta-heurísticas têm surgido na literatura, em particular *frameworks* genéricos em C++ para métodos baseados em pesquisa local. Não existe, no entanto, até ao momento, e segundo o melhor conhecimento do autor, qualquer proposta de abordagem para meta-heurísticas multiobjectivo. Por outro lado, na literatura não é dado ênfase suficiente ao facto de que o paradigma OO se apresenta particularmente apropriado para a implementação e integração, com pesquisa local, de estratégias genéricas de flexibilização como listas de candidatos, vizinhanças variáveis ou hibridização e paralelização de alto nível. Estas considerações desempenham um papel essencial como motivação para o desenvolvimento do *framework* apresentado nesta dissertação.

5
UMA ABORDAGEM ORIENTADA POR OBJECTOS PARA META-HEURÍSTICAS MULTIOBJECTIVO

Neste capítulo propõe-se um *framework* orientado por objectos para meta-heurísticas multiobjectivo que, no seu estado actual de desenvolvimento, foca, em particular, a área da pesquisa local multiobjectivo.

Descreve-se o objectivo do *framework*, e o respectivo desenho detalhado, com estruturas de classes e as respectivas colaborações. Por motivos de espaço e organização do texto, são aqui apresentadas apenas as partes do *framework* relacionadas com a pesquisa local multiobjectivo, sendo as restantes apresentadas no Anexo B.

Na secção 5.1 são expostas as orientações gerais seguidas na descrição do *framework*, que é iniciada, a um nível mais elevado, nas secções 5.2 e 5.3 com, respectivamente, o objectivo e a arquitectura geral do *framework*. Na secção 5.4 faz-se uma apresentação dos *padrões de desenho* utilizados no *framework*. As secções seguintes, 5.5 a 5.10, correspondem às partes do *framework* relacionadas com a pesquisa local multiobjectivo: avaliação de soluções e movimentos, pesquisa local multiobjectivo, extensão para PSA e MOTS*, extensão para vizinhanças variáveis, extensão para estratégias de listas de candidatos e extensão para paralelização e hibridização. Previamente a uma secção de conclusões, é discutida a relação do *framework* com outras abordagens descritas na literatura.

5.1 Introdução

A descrição de *frameworks* tem, por si só, vindo a assumir uma crescente importância, em termos de investigação, na área do desenvolvimento de *software* orientado por objectos (ver, por exemplo, [Mattsson 1996] para uma revisão de várias abordagens).

Refira-se, em especial, uma abordagem recente, designada UML-FW [Fontoura et al. 1999], que consiste numa extensão à linguagem de modelação UML [OMG 1999]. Esta abordagem tem um enorme potencial de utilização, devido à sua associação com a UML, que tem actualmente um elevado nível de disseminação. Um dos casos de aplicação [Fontoura et al. 2000] é, aliás, o *framework Searcher* [Andreatta et al. 1998], apresentado no Capítulo 4, originalmente descrito através de uma adaptação do *template* de descrição de *padrões de desenho* de [Gamma et al. 1995].

Para a descrição do *framework* desenvolvido no âmbito desta dissertação, seguem-se as orientações gerais propostas por [Johnson 1992]. Segundo aquele autor, a documentação de *frameworks* deverá incluir descrições dos seguintes aspectos:

- o objectivo, ou seja, o domínio para o qual foi concebido;
- o desenho detalhado, com estruturas de classes e respectivas colaborações;
- instruções detalhadas sobre a utilização do *framework* para construir aplicações.

O primeiro aspecto merecerá alguma atenção neste capítulo da dissertação, complementando informação apresentada nos capítulos dedicados às meta-heurísticas e às meta-heurísticas multiobjectivo. A análise do domínio é um aspecto fulcral do trabalho desenvolvido, uma vez que entre os modelos de desenvolvimento de *frameworks* referidos no Capítulo 4, aquele que mais naturalmente se integra no trabalho realizado no âmbito da presente dissertação é, exactamente, o que parte da análise do domínio. Esta análise foi já realizada e apresentada, de forma detalhada, em capítulos anteriores, dedicados às meta-heurísticas e, em particular, às meta-heurísticas multiobjectivo.

A descrição do desenho detalhado constituirá a parte mais substancial do presente capítulo. Para a descrição das estruturas de classes e respectivas colaborações serão utilizados dois tipos de diagramas que fazem actualmente parte da UML. No primeiro caso serão utilizados diagramas de classes (*"Static Structure Diagrams"*), que exibem um conjunto de elementos de modelação declarativos (estáticos) como classes, tipos, e o seus conteúdos e relações. No segundo caso utilizar-se-ão diagramas de sequência, que representam as interacções entre objectos, dispostas numa sequência temporal. Em particular, tais diagramas permitem evidenciar os objectos que participam na interacção e a sequência de mensagens

trocadas. No Anexo A apresenta-se um resumo da notação utilizada nestes dois tipos de diagramas.

Os aspectos de utilização do *framework* serão tratados no Capítulo 6, com a sua aplicação a um caso de estudo. Será de notar que algumas das partes do *framework* constituem extensões ao núcleo deste e podem, como tal, ser vistas como casos de utilização. No entanto, dado o carácter estreito da sua integração com o núcleo do *framework*, as respectivas descrições serão realizadas conjuntamente, no âmbito do presente capítulo.

Por motivos de espaço e organização do texto, nem todas as partes do *framework* serão alvo de descrição detalhada neste capítulo. O ênfase será colocado sobre as partes directamente relacionadas com a pesquisa local multiobjectivo, sendo a descrição das restantes remetida para o Anexo B. Na apresentação da arquitectura geral do *framework*, esta estruturação será pormenorizada.

5.2 Objectivo e âmbito do *framework* METHOOD

O *framework* aqui proposto tem a designação de METHOOD (acrónimo para *METaHeuristics Object-Oriented Development*). O seu objectivo é disponibilizar uma arquitectura de base flexível para a construção, aplicação e comparação de meta-heurísticas multiobjectivo. Os contextos em que se propõe a sua utilização são o desenvolvimento de aplicações práticas e a investigação na área da Optimização Combinatória Multiobjectivo, envolvendo meta-heurísticas multiobjectivo.

A principal motivação que levou ao seu desenvolvimento é a convicção de que uma abordagem OO potencia o objectivo referido, das seguintes formas:

- as características básicas do paradigma OO permitem dotar uma tal infra-estrutura, de um elevado grau de flexibilidade e configurabilidade;
- a construção de novas meta-heurísticas ou variantes, através de extensões por composição ou derivação de subclasses, é simplificada e incentivada pela clareza conceptual da infra-estrutura;
- a reutilização dos algoritmos meta-heurísticos disponibilizados permite acelerar e simplificar a aplicação a novos problemas, nomeadamente em cenários de prototipagem e experimentação;
- a definição dos aspectos particulares ao problema num mesmo conjunto de objectos e a partilha de diversos componentes e métodos constitui uma plataforma adequada para comparação de meta-heurísticas.

As principais funcionalidades disponibilizadas pelo METHOOD são as seguintes:

- *template* de pesquisa local multiobjectivo;
- meta-heurísticas multiobjectivo PSA e MOTS*;
- estratégias de listas de candidatos;
- suporte para vizinhanças variáveis;
- uma abordagem de hibridização de alto nível, co-evolutiva e paralelização do tipo *threads* multipesquisa;
- representação dos dados do problema em formato algébrico, com importação a partir de ficheiro de texto.

Tomando como referência o objectivo geral referido, é possível utilizar o *framework* de diversas formas:

- aplicando directamente meta-heurísticas multiobjectivo na resolução de problemas de Optimização Combinatória Multiobjectivo;
- criando extensões com algoritmos básicos que tirem partido da flexibilidade da infra-estrutura de componentes e estratégias elementares, disponibilizados;
- criando extensões com algoritmos compostos que tirem partido, quer da flexibilidade da infra-estrutura de componentes e estratégias elementares, quer de outros algoritmos básicos ou compostos, disponibilizados;
- comparando o desempenho de diversas configurações meta-heurísticas na resolução de um mesmo problema ou conjunto de problemas.

O estado actual de desenvolvimento do *framework* exige, para a sua aplicação, um nível razoável de conhecimento da sua estrutura e funcionamento interno. Esta aplicação é realizada maioritariamente por herança, num esquema de reutilização do tipo *caixa branca*. À medida que se vá ganhando experiência na sua utilização, procurar-se-á consolidar a caracterização do domínio, nomeadamente em relação à definição das suas partes estáveis e flexíveis, e assim permitir a evolução para uma reutilização do tipo *caixa preta*, com a aplicação realizada através de composição.

O ênfase no desenvolvimento do *framework* tem sido colocado nos aspectos de generalidade, procurando-se, no entanto, manter níveis elevados de eficiência. A futura evolução do *framework* passará certamente por uma cuidada análise e melhoria deste compromisso.

5.3 Arquitectura geral

Conforme se referiu no Capítulo 4, em [Fink et al. 1998a] e [Fink et al. 1998b], é sugerido que a questão central no desenvolvimento de *frameworks* para meta-heurísticas consiste na separação entre as características específicas dos problemas e os algoritmos meta-heurísticos. Foi ainda referido que as abordagens encontradas na literatura têm tipicamente realizado esta separação com base em mecanismos de herança ou em polimorfismo estático.

A arquitectura do METHOOD (Figura 5.1) baseia-se na primeira destas opções (herança). O *framework* é maioritariamente constituído pelas partes do domínio não dependentes de aplicação específica. As partes variáveis, que consistem basicamente nos elementos dependentes do problema de optimização a resolver, são consideradas através de classes abstractas. Para a resolução de problemas particulares, estas partes são configuráveis através da utilização de herança.

A base do *framework* é constituída por estruturas e colaborações básicas, maioritariamente relativas a componentes e estratégias elementares de meta-heurísticas. Tal inclui parte da vertente de definição do problema - as classes abstractas correspondentes aos elementos variáveis do domínio: soluções, incrementos, movimentos e funções de avaliação. A outra parte desta vertente, relativa ao suporte para dados do problema, é constituída por classes para armazenamento dos dados do problema, baseadas em modelação algébrica.

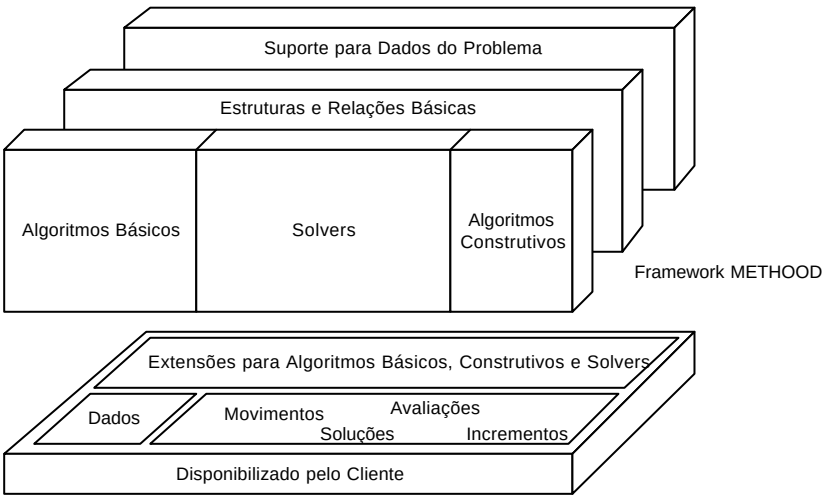


Figura 5.1 Arquitectura geral do *framework* METHOOD

Na parte relativa às técnicas heurísticas, o *framework* apresenta:

- um *template* de algoritmos construtivos;
- uma concretização do *template* de algoritmos construtivos;
- um *template* de pesquisa local multiobjectivo;
- duas concretizações do *template* de pesquisa local multiobjectivo, implementando algoritmos básicos: as meta-heurísticas PSA e MOTS*;
- variantes destes algoritmos básicos, envolvendo vizinhanças variáveis e estratégias de listas de candidatos;
- *solvers* meta-heurísticos: algoritmos de nível superior aos anteriores, que permitem articular, por um lado, os algoritmos construtivos e os algoritmos básicos de pesquisa local, e, por outro, diversos algoritmos básicos de pesquisa local em abordagens de hibridização e paralelização de alto nível.

Assume ainda particular relevância, ao nível da arquitectura geral do *framework*, o facto de este utilizar duas bibliotecas de estruturas de dados e algoritmos: a *Standard Template Library* (STL) [Musser, Saini 1996] e a *Library of Efficient Data Structures and Algorithms* (LEDA) [Mehlhorn, Näher 1995].

A estrutura da arquitectura geral orientará a apresentação mais detalhada do *framework*, precedida de uma breve descrição dos *padrões de desenho* utilizados.

Como referido atrás, e para efeitos de clareza do texto, a descrição do desenho detalhado do *framework* será dividida entre o presente capítulo e o Anexo B. Neste capítulo, são descritas as partes que mais directamente se relacionam com a pesquisa local multiobjectivo:

1. avaliação multiobjectivo de soluções e movimentos;
2. algoritmos de pesquisa local multiobjectivo;
3. PSA e MOTS*;
4. suporte para vizinhanças variáveis e estratégias de listas de candidatos;
5. suporte para abordagens de paralelização e hibridização.

No Anexo B, serão focados os seguintes aspectos:

1. bibliotecas de estruturas de dados e algoritmos utilizadas;
2. interface entre cliente e *framework*;

3. suporte para os dados do problema, baseado em modelação algébrica;
4. *solvers* meta-heurísticos;
5. algoritmos construtivos.

A descrição das estruturas e colaborações base será distribuída pelos diversos pontos, permitindo contextualizá-las de forma mais adequada e evitar repetição de informação. Confere-se destaque apenas à parte relativa à avaliação multiobjectivo de soluções e movimentos, devido à dimensão e complexidade que esta envolve, em termos práticos.

5.4 *Padrões de desenho* utilizados

É feita nesta subsecção uma descrição sucinta dos *padrões de desenho* utilizados no *framework*. Todos estes *padrões* pertencem ao "catálogo" de [Gamma et al. 1995], onde poderá ser encontrada a sua descrição mais pormenorizada.

O *padrão* Estratégia (*"Strategy"*), apresentado em detalhe no Capítulo 4, define uma família de algoritmos, encapsula-os e torna-os intermutáveis. Desta forma, permite variar o algoritmo a utilizar de forma independente do cliente.

O *padrão* Método *Template* (*"Template Method"*) define, numa operação, o esqueleto de um algoritmo, passando para um conjunto de subclasses a redefinição de alguns dos passos do algoritmo, sem alterar a estrutura deste. Tal é conseguido através da definição do algoritmo em termos de operações abstractas que, ao serem implementadas pelas subclasses, disponibilizam um comportamento concreto.

O *padrão* Iterador (*"Iterator"*) permite disponibilizar um acesso sequencial aos elementos de um objecto agregado, sem expor a representação a este subjacente. Tal é conseguido retirando ao objecto agregado a responsabilidade pelo acesso sequencial, e colocando-a num objecto iterador. Esta classe define um interface para aceder aos elementos do objecto agregado.

O *padrão* Protótipo (*"Prototype"*) possibilita a especificação de tipos de objectos a criar através de uma instância protótipo, criando novos objectos por cópia deste protótipo.

O *padrão* Estado (*"State"*) permite a um objecto alterar o seu comportamento quando se altera o seu estado interno, simulando uma mudança de classe. Tal é conseguido através da localização, numa classe abstracta, de um interface comum a diversas subclasses concretas, que representarão diferentes comportamentos. O interface relevante é definido numa outra classe, que delega na classe de comportamento concreta actual os pedidos relacionados com comportamento. A transição de estados/comportamentos poderá ser controlada pelas subclasses concretas ou pela própria classe de interface.

5.5 Avaliação de soluções e movimentos

Para a avaliação de soluções e movimentos num contexto multiobjectivo, o *framework* disponibiliza um vector de critérios, associado a um conjunto de operações mais frequentes. Uma vez que as diversas meta-heurísticas multiobjectivo utilizam vectores de pesos e vectores de pesos equalizados, em estreita ligação com os vectores de critérios, estes são disponibilizados conjuntamente no *framework*.

A cada critério estará associada um função de avaliação que, no caso das soluções, fará uma avaliação global, enquanto no caso dos movimentos, se baseará na variação de avaliação introduzida pela respectiva execução sobre uma dada solução.

Estrutura e participantes

Na Figura 5.2 apresenta-se a estrutura de classes desta parte do *framework*.

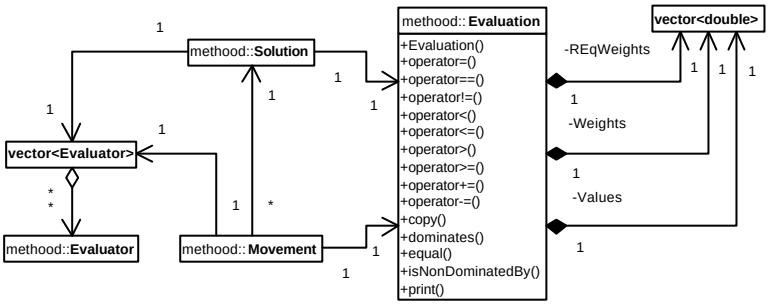


Figura 5.2 Diagrama de classes para a avaliação de soluções e movimentos

Os participantes são os seguintes:

- *Solução* (*Solution*). Contém a representação da solução. Cada solução tem uma avaliação (*Evaluation*) e tem associado um conjunto de funções de avaliação (*Evaluator*), uma para cada um dos objectivos considerados.
- *Movimento* (*Movement*). Contém a representação de um movimento sobre uma solução. Cada movimento tem uma avaliação (*Evaluation*), correspondente à sua execução sobre uma solução particular (*Solution*), e tem associado um conjunto de funções de avaliação (*Evaluator*), uma para cada um dos objectivos considerados. A consideração deste novo conjunto de funções de avaliação é motivada pelo impacto tipicamente reduzido da execução de um movimento sobre uma solução. Por conseguinte, será normalmente mais eficiente avaliar apenas as variações introduzidas nos valores das funções objectivo pela execução do movimento.

- *Função de avaliação* (*Evaluator*). Avalia uma solução ou o impacto de um movimento sobre uma solução segundo um determinado critério.
- *Avaliação* (*Evaluation*). Contém um vector de valores (*Values*) correspondentes à avaliação segundo cada um dos objectivos, um vector de pesos (*Weights*) e um vector de pesos equalizados (*REqWeights*). Disponibiliza as operações mais comuns sobre vectores de critérios.

Colaborações

Neste âmbito, há a considerar dois cenários de colaboração:

1. *Avaliação de movimentos* (Figura 5.3).

   Quando é invocada a avaliação de um movimento, a primeira operação, *UpdateEvalInfo()*, abstracta ao nível do movimento, deverá ser executada pelas classes derivadas com o objectivo de actualizar um eventual interface de informação entre estas e os avaliadores.

   Um iterador de valores permite percorrer o vector de valores de avaliação, preenchendo-o com os valores calculados pelas funções de avaliação correspondentes. Um iterador de avaliadores permite percorrer, em simultâneo, o vector de avaliadores.

   A avaliação do movimento é obtida através da adição da variação calculada à avaliação da solução considerada.

2. *Avaliação de soluções* (Figura 5.4).

   A colaboração processa-se de forma idêntica à *avaliação de movimentos*, a menos da obtenção do valor da avaliação do movimento.

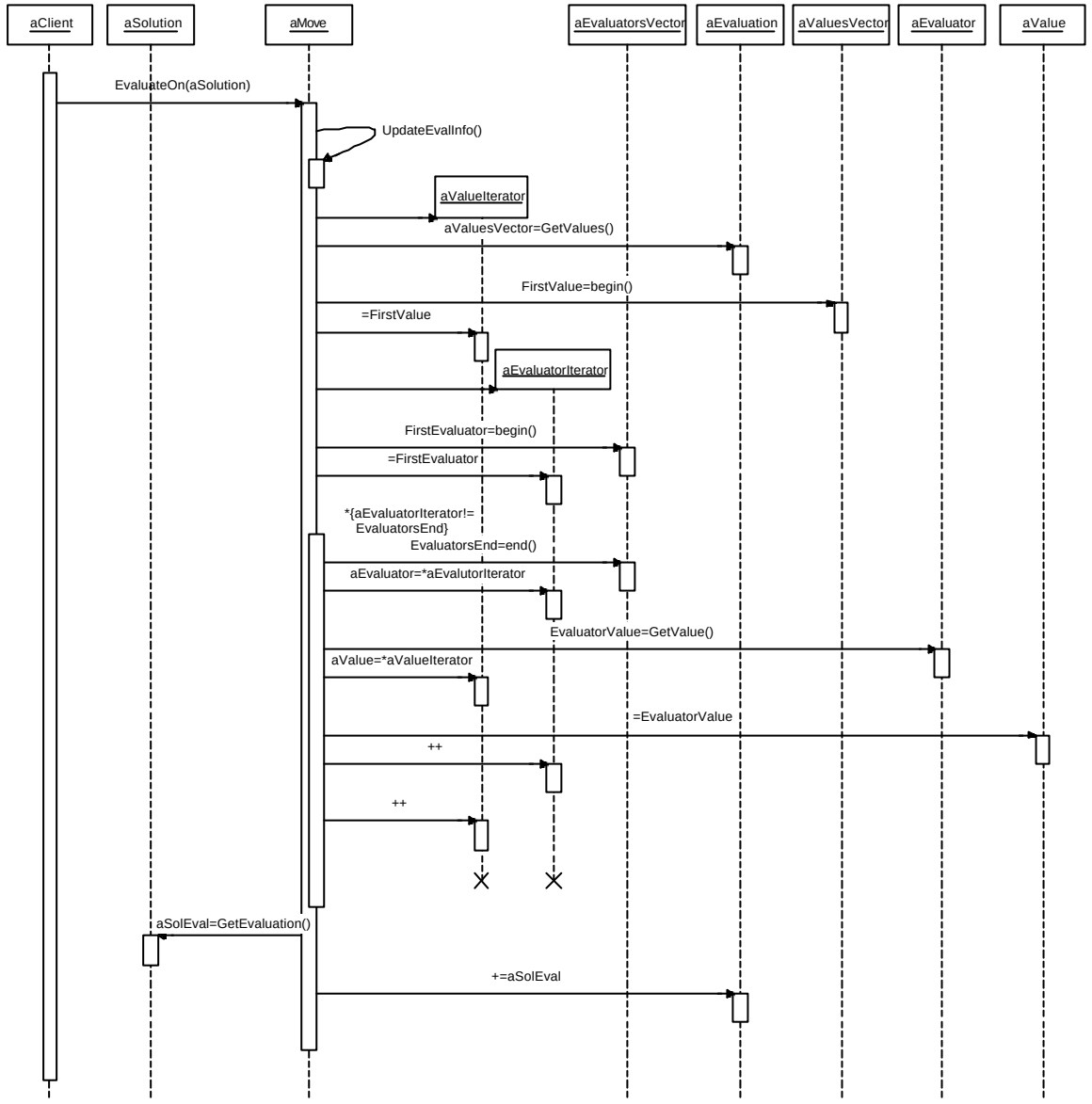


Figura 5.3 Diagrama de sequência para a avaliação de movimentos

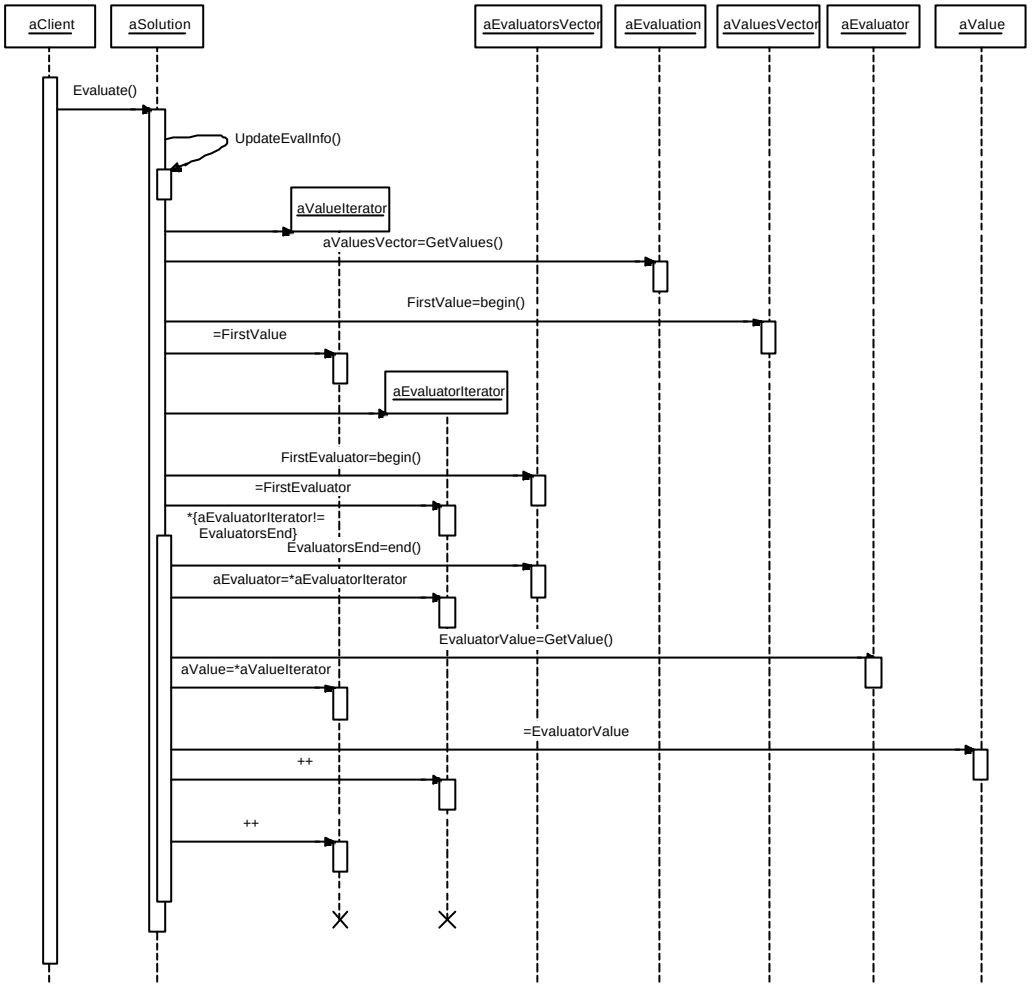


Figura 5.4 Diagrama de sequência para a avaliação de soluções

5.6 Pesquisa local multiobjectivo

Esta parte do *framework* baseia-se na aplicação de quatro *padrões de desenho* constantes do "catálogo" de [Gamma et al. 1995]:

- O *padrão* Método *Template* é aplicado em duas situações:
  - A classe *pesquisa local multiobjectivo* é derivada de uma classe abstracta mais geral que constitui um *template* para processos iterativos. A disponibilização da classe *processo iterativo* tem como intenção encorajar a estruturação dos diversos aspectos envolvidos em processos iterativos, quando estes apresentam elevada dimensão ou complexidade lógica. Não se preconiza a utilização desta classe para pequenos ciclos, em que se atinge uma eficiência superior com estruturas próprias da linguagem de programação.
  - A *pesquisa local multiobjectivo* define os passos invariantes de um algoritmo de pesquisa local multiobjectivo, sendo responsabilidade de classes dela

derivadas, definir as operações primitivas abstractas que concretizam um algoritmo particular, como o PSA ou a MOTS*. Esta aplicação do *padrão* Método *Template* será descrita em maior pormenor adiante, com a apresentação detalhada dos algoritmos referidos.

- O *padrão* Iterador. Uma vizinhança ou um *gerador de movimentos* (cuja descrição é apresentada adiante) podem ser vistos como sendo constituídos por movimentos. A utilização de um iterador para acesso (sequencial) aos movimentos evita a necessidade de expor as representação internas utilizadas naquelas classes.

  A aplicação deste *padrão* é realizada com as seguintes adaptações:

  – A vizinhança delega no gerador de movimentos os pedidos do iterador. Esta delegação está relacionada com o mecanismo de suporte para vizinhanças variáveis, que recorre à variação do gerador de movimentos.

  – O gerador de movimentos é um *contentor-gerador*, no sentido em que só é criada uma instância de um movimento quando ocorre um pedido desse movimento. Pretende-se, desta forma, alcançar um melhor compromisso de utilização de memória, uma vez que os movimentos são gerados apenas quando necessário. Esta é também uma forma simples de conferir independência dos movimentos em relação ao gerador, uma vez que, logo após a sua geração, o seu ciclo de vida passa a ser totalmente independente do gerador.

- Uma aplicação do *padrão* Estratégia resulta no estabelecimento de *famílias de algoritmos de definição de pesos*, dos quais a pesquisa local será cliente: estas estratégias corresponderão às definições de pesos implementadas no PSA e na MOTS*.

- O *padrão* Protótipo é aplicado em duas situações:

  – A cada gerador de movimentos está associado um protótipo de movimento, cujos serviços são utilizados para criar novos movimentos.

  – Um protótipo de vizinhança permite ao algoritmo de pesquisa local a criação de novas vizinhanças para cada uma das soluções que constituem a sua população inicial.

Apresentam-se de seguida a estrutura e colaborações de classes presentes nesta parte do *framework*. Por questões de clareza de exposição, a apresentação da aplicação do *padrão* Método *Template* relativamente ao processo iterativo é destacada da restante apresentação geral.

Estrutura e participantes

O diagrama de classes apresentado na Figura 5.5 apresenta a estrutura de classes geral para a pesquisa local multiobjectivo.

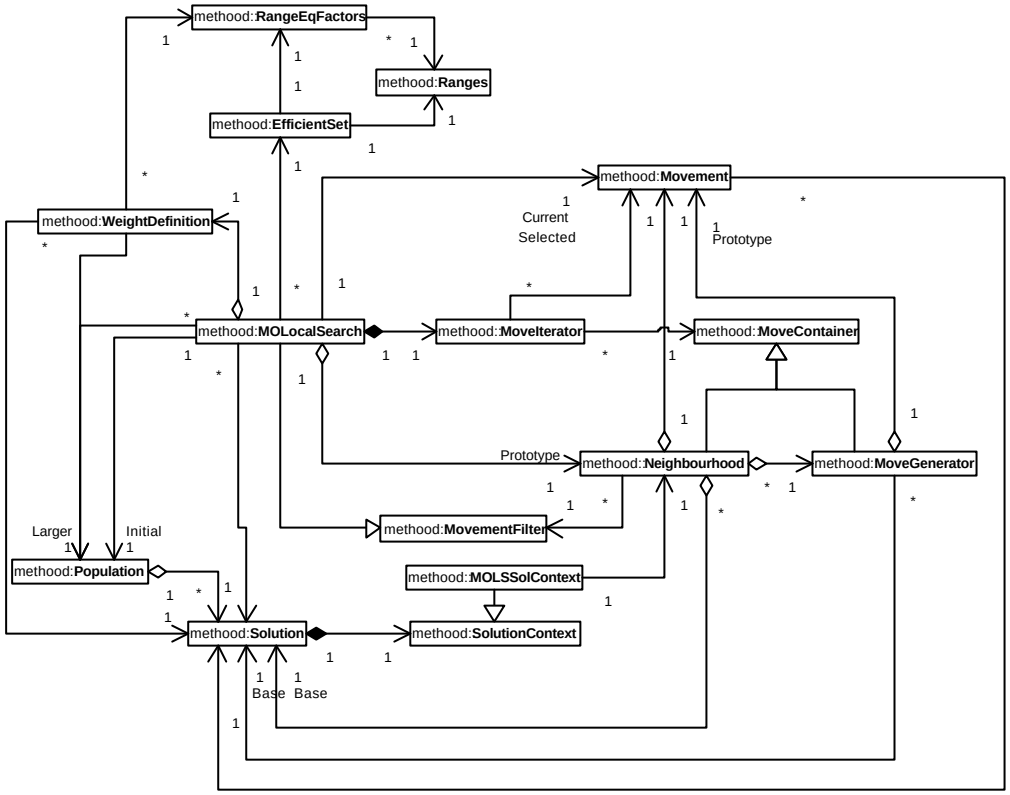


Figura 5.5 Diagrama de classes para pesquisa local multiobjectivo

Os participantes nesta parte do *framework* são:

- *Pesquisa local multiobjectivo* (*MOLocalSearch*). Implementa a pesquisa local multiobjectivo que, de acordo com um princípio meta-heurístico concreto, itera sobre uma população de soluções ($Population_{Initial}$), construindo uma aproximação ao conjunto de soluções eficientes (*EfficientSet*) do problema em causa.

  Um protótipo de vizinhança ($Neighbourhood_{Prototype}$) permitirá construir uma vizinhança para cada uma das soluções da população inicial.

  A selecção dos movimentos é realizada de acordo com um esquema de pesos, definidos por uma estratégia de definição de pesos (*WeightDefinition*). Esta estratégia tem em consideração as soluções pertencentes a uma população alargada ($Population_{Larger}$). Uma definição adequada destes pesos permitirá aproximar a população da fronteira eficiente e dispersá-la sobre essa mesma fronteira.

O algoritmo pode ainda ser visto como um filtro de movimentos (*MovementFilter*), no sentido em que o princípio meta-heurístico que implementa pode ser redutor de vizinhanças (por exemplo, em TS não são elegíveis os movimentos tabu que não satisfaçam o critério de aspiração).

- *População* (*Population*). Implementa uma lista de soluções. Como população inicial ($Population_{Initial}$), essas soluções irão ser trabalhadas pelos algoritmos de pesquisa local multiobjectivo. A população alargada ($Population_{Larger}$) será composta pelas soluções da população inicial e outras soluções de referência, que irão ser utilizadas no âmbito das estratégias de definição de pesos.
- *Solução* (*Solution*).
- *Definição de pesos* (*WeightDefinition*). Implementa o interface que será usado por estratégias concretas de definição de pesos.
- *Contexto de solução* (*SolutionContext*). Classe abstracta que permite definir em classes derivadas, elementos específicos a cada solução.
- *Contexto de solução para pesquisa local multiobjectivo* (*MOLSSolContex*). Classe derivada do contexto de solução. Contém a vizinhança da solução.
- *Vizinhança* (*Neighbourhood*). Incorpora o conceito de vizinhança, delegando ao gerador de movimentos (*MoveGenerator*) com que é configurada, os pedidos de movimentos que permitirão construir todas as suas soluções. Como protótipo ($Neighbourhood_{Prototype}$), será utilizada para criar novas vizinhanças para todas as soluções da população inicial.
- *Filtro de movimentos* (*MoveFilter*). Implementa a acção do princípio meta-heurístico, como filtro de movimentos.
- *Gerador de movimentos* (*MoveGenerator*). Cria movimentos, a partir de um protótipo de movimento ($Movement_{Prototype}$). É utilizado para a construção de vizinhanças.
- *Contentor de movimentos* (*MoveContainer*). Define o interface que um contentor concreto deve disponibilizar para acesso através de um iterador.

- *Iterador de movimentos* (*MoveIterator*). Implementa o acesso sequencial aos elementos de um contentor de movimentos e mantém a posição actual da travessia do contentor.

- *Movimento* (*Movement*). Contém a representação de um movimento sobre a solução. Como protótipo ($Movement_{Prototype}$), permite criar novos movimentos a serem configurados pelo gerador de movimentos. Como movimento seleccionado ($Movement_{Selected}$), constitui o elemento da vizinhança seleccionado para execução sobre a solução actual. Como movimento actual ($Movement_{Current}$) corresponde aos diversos movimentos da vizinhança da solução actual.

- *Aproximação ao conjunto de soluções eficientes* (*EfficientSet*). Conterá o resultado da execução do algoritmo, que consistirá nas soluções eficientes do conjunto de soluções por ele pesquisadas. Na literatura são apontadas implementações bastante eficientes para este componente, com base em *quadtrees*. No entanto, a implementação de um componente genérico para qualquer número de objectivos, reveste-se de uma complexidade não enquadrável no contexto deste trabalho. Assim, a implementação assegurada actualmente no *framework* é baseada numa estrutura de dados menos eficiente, mas mais simples: a lista ligada.

  É considerada a possibilidade de definir dois tipos de limiares inferiores para a inserção de uma solução na aproximação ao conjunto de soluções eficientes: no espaço de soluções, uma distância mínima a qualquer outra solução com idêntica avaliação; no espaço dos objectivos, uma distância mínima a qualquer outra solução.

- *Factores de equalização de gamas* (*RangeEqFactors*). Implementa os factores de equalização de gamas, tal como definidos no capítulo dedicado às meta-heurísticas multiobjectivo.

- *Gamas* (*Ranges*). Implementa as gamas dos diferentes objectivos na aproximação ao conjunto de soluções eficientes.

O diagrama de classes apresentado na Figura 5.6 apresenta a estrutura de classes para a aplicação do *padrão* Método *Template* em pesquisa local multiobjectivo.

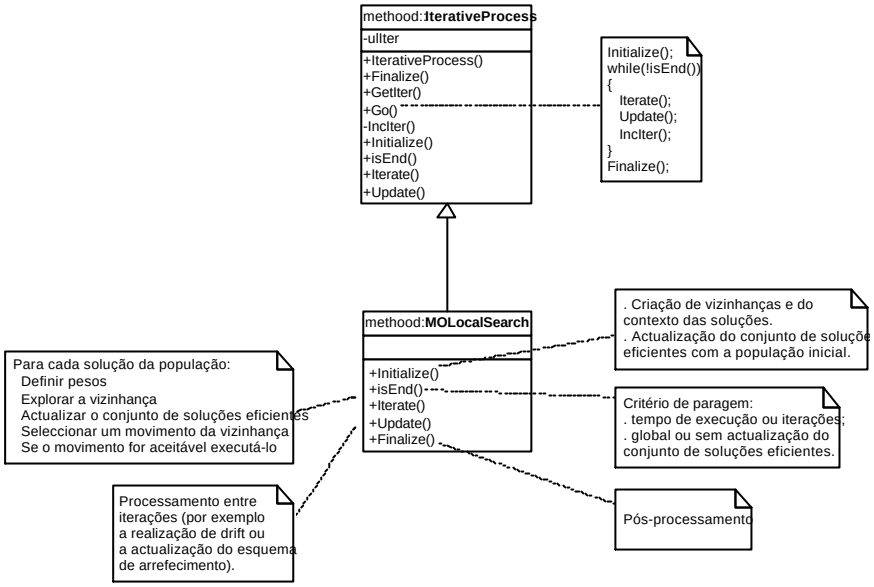


Figura 5.6 Diagrama de classes para a aplicação do padrão Método *Template* em pesquisa local multiobjectivo

Os participantes nesta parte do *framework* são:

- *Processo Iterativo* (*IterativeProcess*). Define as operações primitivas abstractas próprias de um processo iterativo, que subclasses concretas deverão definir para implementar os passos de um algoritmo iterativo: inicialização (*Initialize*), critério de paragem (*isEnd*), iteração (*Iterate*), actualização de dados e informação de controlo (*Update*) e finalização (*Finalize*). Implementa um método *template* que define o esqueleto de um algoritmo iterativo (*Go*). Este método utiliza as operações primitivas, bem como outras definidas na própria classe.
- *Pesquisa local multiobjectivo* (*MOLocalSearch*). Implementada como um processo iterativo, define as operações primitivas enumeradas na classe anterior, de acordo com as anotações da Figura 5.6. Inclui assim, além da iteração com a exploração das vizinhanças das soluções, suporte explícito para a inicialização dos respectivos contextos, para critérios de paragem, processamento entre iterações e pós-processamento.

Colaborações

Na aplicação do *padrão* Método *Template*, a colaboração consiste apenas no facto de o algoritmo de pesquisa local se basear no processo iterativo para implementar um conjunto de passos invariantes. Também para o *padrão* Protótipo, a colaboração se reveste de simplicidade: no caso dos movimentos, o gerador de movimentos solicita ao movimento protótipo a

criação de um novo movimento; no caso das vizinhanças, o algoritmo de pesquisa local solicita à vizinhança protótipo, a criação de uma nova vizinhança.

Adicionalmente, distinguem-se quatro cenários de colaboração relevantes:

1. *Inicialização* (Figura 5.7). Criação e configuração das vizinhanças e actualização, com a população inicial, da aproximação ao conjunto de soluções eficientes.
2. *Actualização da aproximação ao conjunto de soluções eficientes* (Figura 5.8). Além da actualização da aproximação, envolve a actualização das gamas e factores de equalização de gamas.
3. *Pesquisa local* propriamente dita (Figura 5.9). Assente na iteração sobre uma população de soluções, compreende as seguintes fases: definição de pesos (*WEIGHT DEFINITION*), preparação da vizinhança (*NEIGHBOURHOOD SETUP*), pesquisa (*SEARCH*), selecção de movimento (*SELECTION*) e execução do movimento seleccionado (*EXECUTION*).
4. *Colaboração entre iterador, vizinhança e gerador de movimentos* (Figura 5.10). Este cenário de colaboração, embora parte integrante do anterior, é destacado por motivos de clareza de exposição.

No cenário de inicialização, a população inicial é percorrida, através de um iterador de populações, para inicialização do contexto das respectivas soluções: para cada solução é realizada a avaliação e é criada uma vizinhança, recorrendo aos serviços do protótipo de vizinhança; é ainda criado um contexto, caso ainda não exista, que é parametrizado com a vizinhança criada; esta vizinhança, por sua vez, é parametrizada com a solução actual e com um filtro de movimentos, que corresponde ao algoritmo de pesquisa local multiobjectivo.

Na actualização da aproximação ao conjunto de soluções eficientes (que poderá ser solicitada para uma população, uma solução ou um movimento sobre uma solução), é sempre desencadeada a actualização das gamas e dos factores de equalização de gamas.

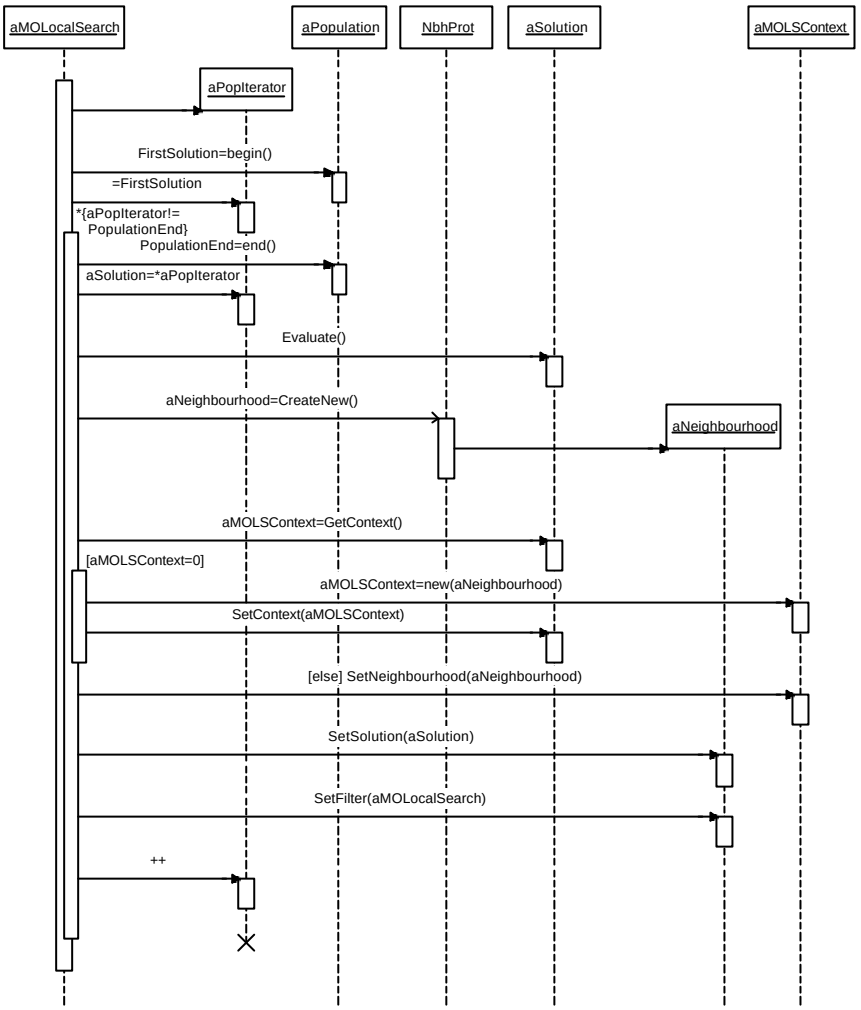


Figura 5.7 Diagrama de sequência para inicialização em pesquisa local multiobjectivo

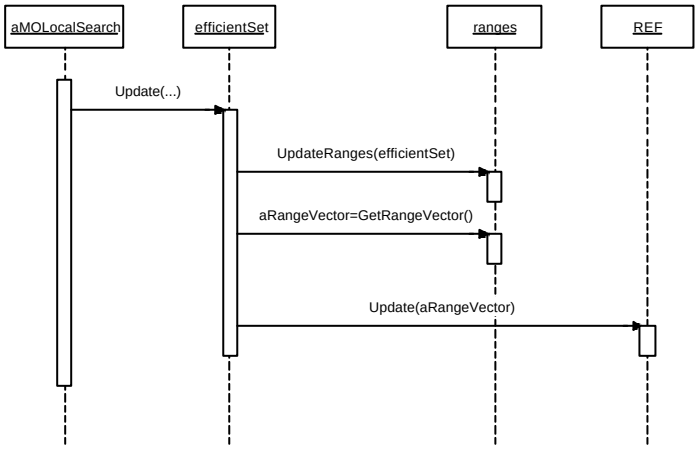


Figura 5.8 Diagrama de sequência para actualização da aproximação ao conjunto de soluções eficientes

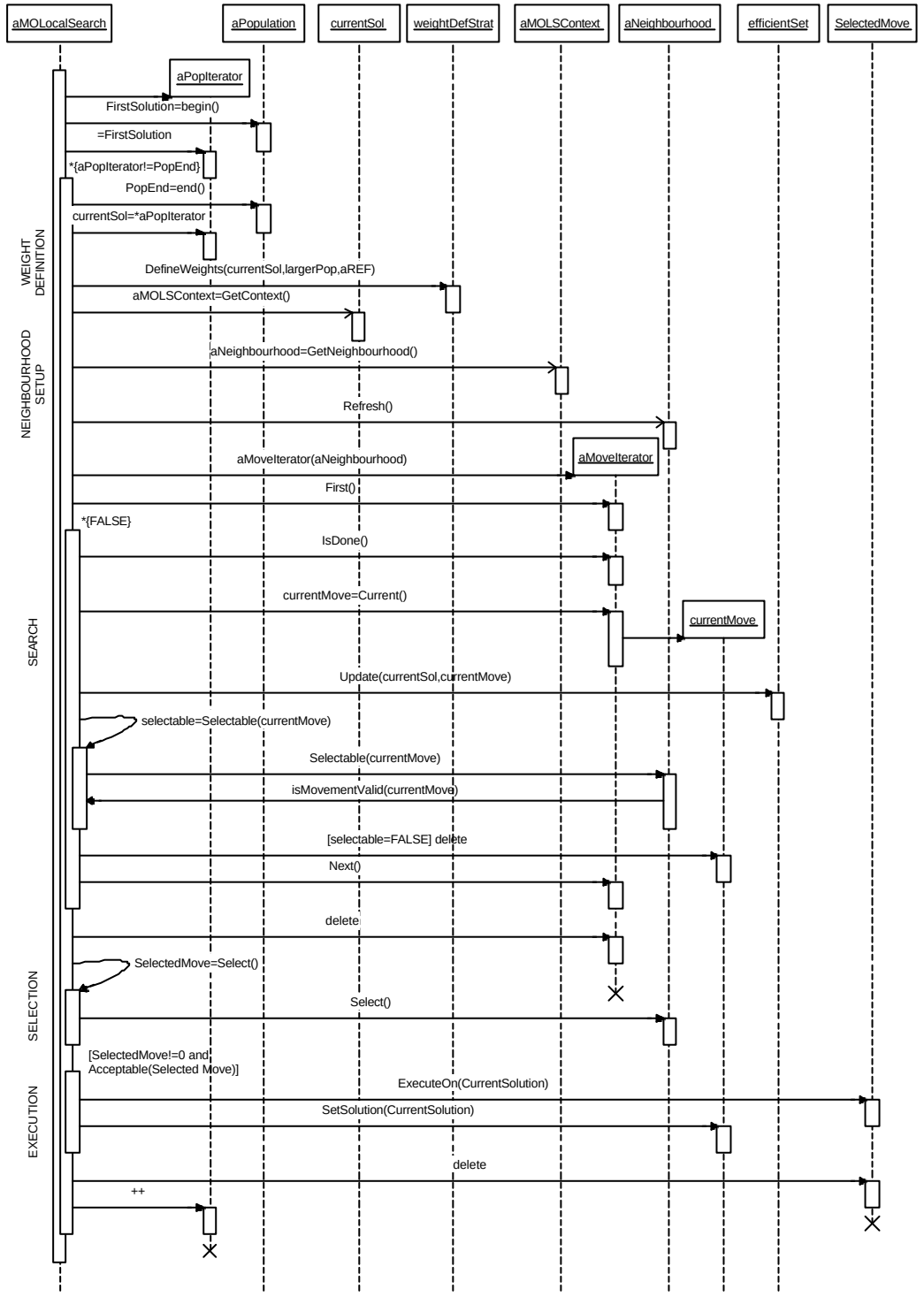


Figura 5.9 Diagrama de sequência para a pesquisa local multiobjectivo

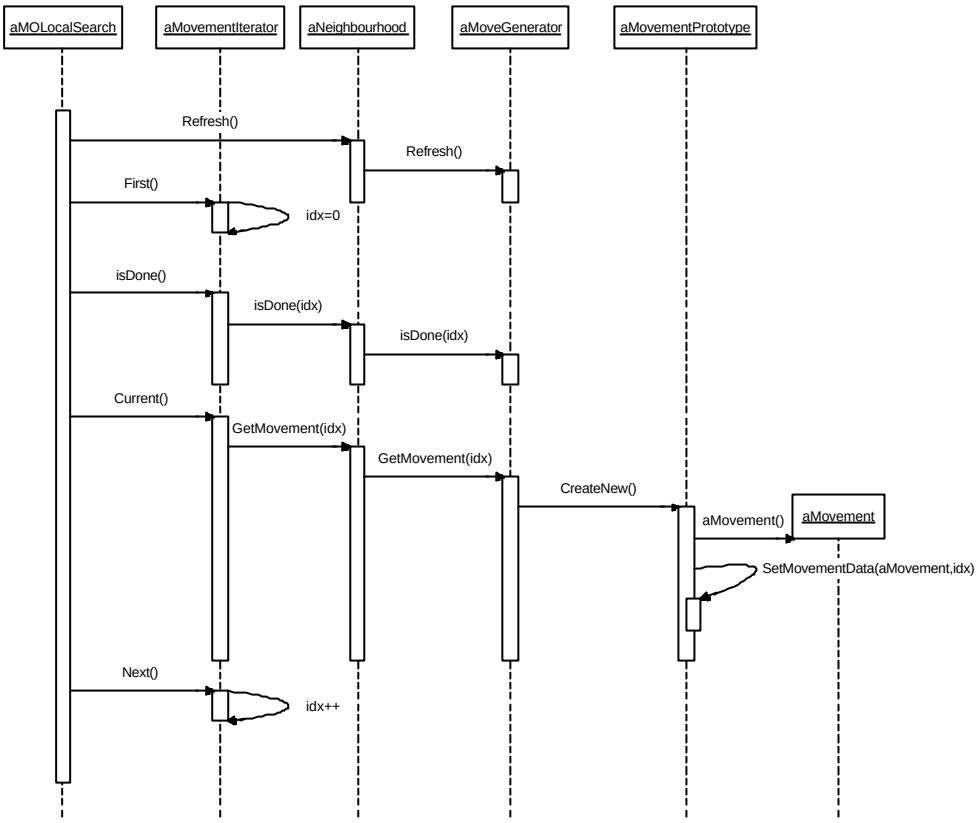


Figura 5.10 Diagrama de sequência para a colaboração entre iterador, vizinhança e gerador de movimentos

A pesquisa local multiobjectivo compreende as seguintes fases de colaboração:

1. *Definição de pesos* (*WEIGHT DEFINITION*). A estratégia de definição de pesos com que o algoritmo foi parametrizado é utilizada para definir os pesos da solução actual. Para tal, deverá conhecer a solução actual, a população alargada e os factores de equalização de gamas.

2. *Preparação da vizinhança* (*NEIGHBOURHOOD SETUP*). Consiste na recuperação da vizinhança da solução actual, a partir do respectivo contexto e no seu *refresh*. O serviço de *refresh* destina-se a permitir actualizações internas de informação da vizinhança no reinício da sua utilização.

3. *Pesquisa* (*SEARCH*). A obtenção dos movimentos que fazem parte da vizinhança é realizada através de um iterador de movimentos. Para cada movimento, é desencadeada a actualização da aproximação ao conjunto de soluções eficientes e é verificada a sua elegibilidade segundo critérios da própria vizinhança e da meta-heurística, funcionando como filtro de movimentos.

De entre os movimentos considerados na vizinhança, é seleccionado um para execução. O movimento seleccionado é, no caso da vizinhança básica, aquele que apresentar um melhor valor do produto escalar entre os respectivos valores dos critérios e os pesos equalizados da solução actual.

4. *Execução de movimento* (EXECUTION). O movimento seleccionado é executado sobre a solução e eliminado. A alteração da solução é sinalizada à vizinhança.

Na colaboração entre iterador, vizinhança e gerador de movimentos, o algoritmo de pesquisa local solicita, inicialmente, ao gerador de movimentos, um *refresh*. Este serviço destina-se a permitir actualizações internas de informação, do gerador de movimentos, como, por exemplo, a geração de um novo conjunto aleatório de movimentos, no caso de geradores aleatórios.

Através do iterador, serão percorridos todos os movimentos, sendo a sequência de cada movimento identificada por um índice. À obtenção de cada movimento, está associada a criação de uma instância desse movimento, realizada com recurso aos serviços do protótipo de movimento, que cria um movimento e o disponibiliza ao gerador para uma configuração adequada.

5.7 Extensão para PSA e MOTS*

A extensão do núcleo do *framework* para construir meta-heurísticas concretas é realizada com base na aplicação do *padrão* Método *Template*, seguindo várias abordagens da literatura [Vaessens et al. 1995, Andreatta et al. 1998, Schaerf et al. 1999]. É com base na aplicação deste *padrão*, que se estrutura a apresentação desta parte do *framework*.

Por forma a permitir uma melhor organização da exposição, confere-se destaque à descrição da lista tabu, utilizada pela MOTS*, que envolve alguma complexidade, e em cuja concepção são aplicados dois *padrões de desenho*: o *padrão* Iterador, para o acesso aos movimentos que fazem parte da lista, e o *padrão* Protótipo, para a criação de novas listas tabu, para as soluções da população inicial.

Estrutura e participantes

O diagrama de classes apresentado na Figura 5.11 apresenta a estrutura de classes geral para as duas meta-heurísticas concretas, enquanto o diagrama de classes da Figura 5.12 apresenta a estrutura de classes relacionada com a lista tabu.

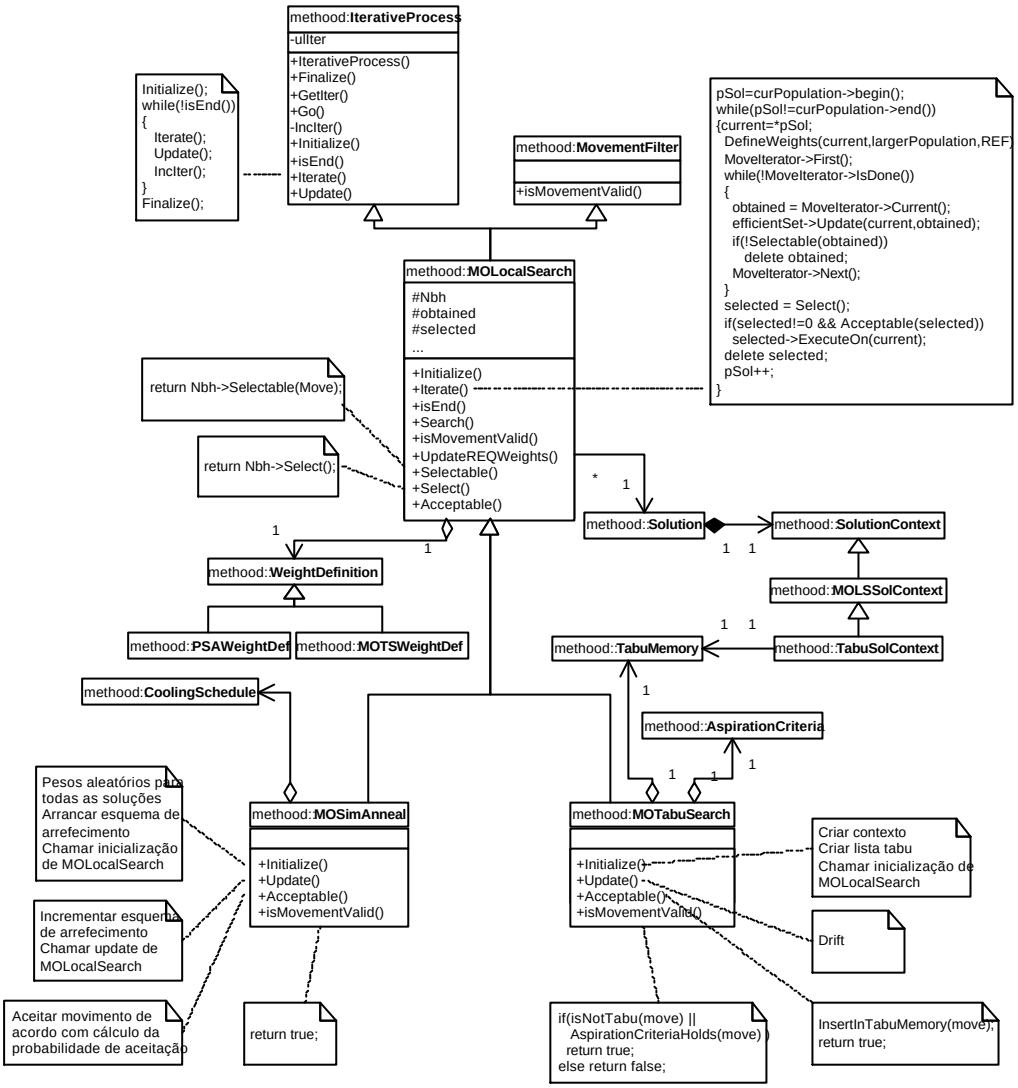


Figura 5.11 Diagrama de classes para as duas meta-heurísticas concretas

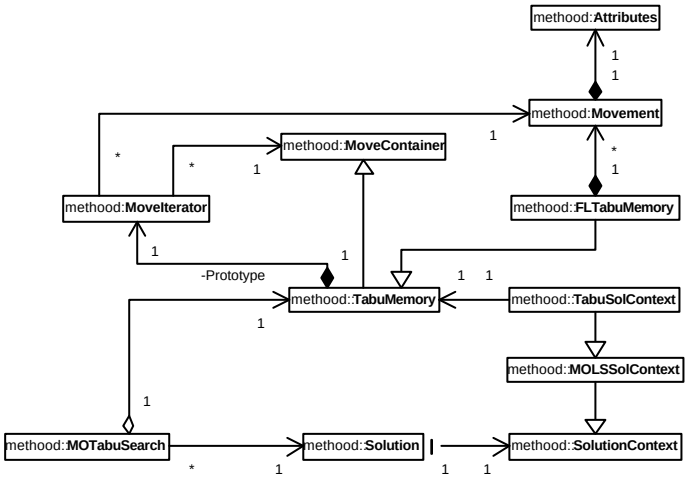


Figura 5.12 Diagrama de classes para a lista tabu

Os participantes nesta parte do *framework* são:

- *Processo iterativo* (*IterativeProcess*).
- *Pesquisa local multiobjectivo* (*MOLocalSearch*).
- *PSA* (*MOSimAnneal*). Implementado como um processo iterativo, define as operações primitivas enumeradas em *MOLocalSearch*, de acordo com as anotações da Figura 5.11: arranca e actualiza um esquema de arrefecimento (*CoolingSchedule*), aplica uma probabilidade de aceitação e considera elegíveis para execução, todos os movimentos gerados na vizinhança.
- *MOTS** (*MOTabuSearch*). Implementado como um processo iterativo, de forma idêntica a *MOSimAnneal*. Em particular, cria e actualiza, para cada solução, uma memória tabu (*TabuMemory*) de movimentos, implementa uma estratégia de *"drift"*, e estabelece um critério de aceitação de movimentos e um critério de aspiração (*AspirationCriteria*).
- *Esquema de arrefecimento* (*CoolingSchedule*). Implementa um esquema de arrefecimento, com os componentes *standard*: temperatura inicial, comprimento do *"plateau"*, factor de arrefecimento e temperatura final.
- *Critério de aspiração* (*AspirationCriteria*). Implementa um critério de aspiração, tendo em conta o movimento actual e a memória tabu: o movimento actual deve dominar todos os movimentos da lista tabu, ou apresentar, em relação a estes, um melhor valor do produto escalar dos pesos equalizados da solução actual, pelos respectivos valores dos critérios.
- *Memória tabu* (*TabuMemory*). Define o interface para uma memória tabu e é derivada do contentor de movimentos (*MoveContainer*). A classe concreta disponibilizada é baseada numa lista ligada (*FLTabuMemory*).
- *Solução* (*Solution*).
- *Contexto de solução* (*SolutionContext*).
- *Contexto de pesquisa local multiobjectivo* (*MOLSSolContext*).
- *Contexto de TS* (*TabuSolContext*). Classe derivada do contexto de pesquisa local multiobjectivo, acrescentando-lhe a memória tabu que corresponderá a cada solução.
- *Lista tabu* (*FLTabuMemory*). Concretização da classe memória tabu, que se baseia numa lista de movimentos de tamanho fixo.

- *Contentor de movimentos* (*MoveContainer*).
- *Iterador de movimentos* (*MoveIterator*).
- *Movimento* (*Movement*).
- *Atributos* (*Attributes*). Implementa os atributos de um movimento, bem como eventuais atributos de uma solução particular em que o movimento tenha sido executado. É através destes atributos que se verifica o estado tabu de um movimento.
- *Filtro de movimentos* (*MovementFilter*).
- *Definição de pesos* (*WeightDefinition*).
- *Definição de pesos do PSA* (*PSAWeightDef*). Implementa a estratégia de definição de pesos do PSA.
- *Definição de pesos da MOTS** (*MOTSWeightDef*). Implementa a estratégia de definição de pesos da MOTS*.

Colaborações

Tal como no caso da pesquisa local multiobjectivo, também nestas extensões a colaboração na aplicação do *padrão* Método *Template* consiste apenas no facto de os algoritmos concretos se basearem no *template* de pesquisa local multiobjectivo para implementar o conjunto de passos invariantes. No caso do *padrão* Protótipo, será o algoritmo MOTS* a solicitar à lista tabu protótipo a criação de novas listas tabu.

Relativamente à utilização da memória tabu, a colaboração estrutura-se em duas partes:

1. *Inserção de movimento* (Figura 5.13). É criada uma cópia do movimento, que é inserida na cauda da lista. Se o comprimento da lista exceder um valor pré-definido, é retirado o elemento à cabeça da lista.
2. *Verificação do estado tabu* (Figura 5.14). É utilizado um iterador de movimentos para percorrer a lista tabu até ao final desta ou ser encontrado um movimento que torne tabu o movimento actual.

   A verificação do estado tabu é delegada nos atributos do movimento actual e do movimento da lista tabu em análise. Esta verificação implica que entre os atributos haja compatibilidade de tipos (ou seja, deverão pertencer à mesma classe, ou os atributos do movimento da lista tabu deverão ser herdados dos

atributos do movimento actual). No caso de os atributos não serem compatíveis, o movimento actual não é considerado tabu.

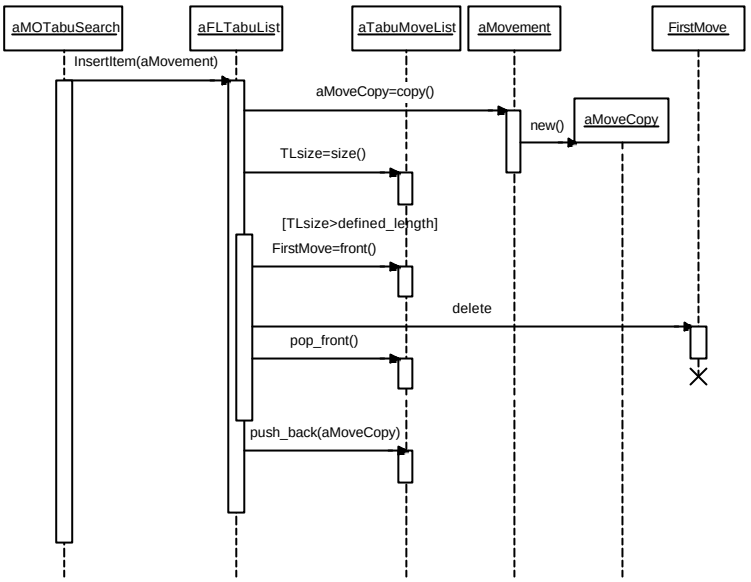


Figura 5.13 Diagrama de sequência para inserção de movimentos na memória tabu

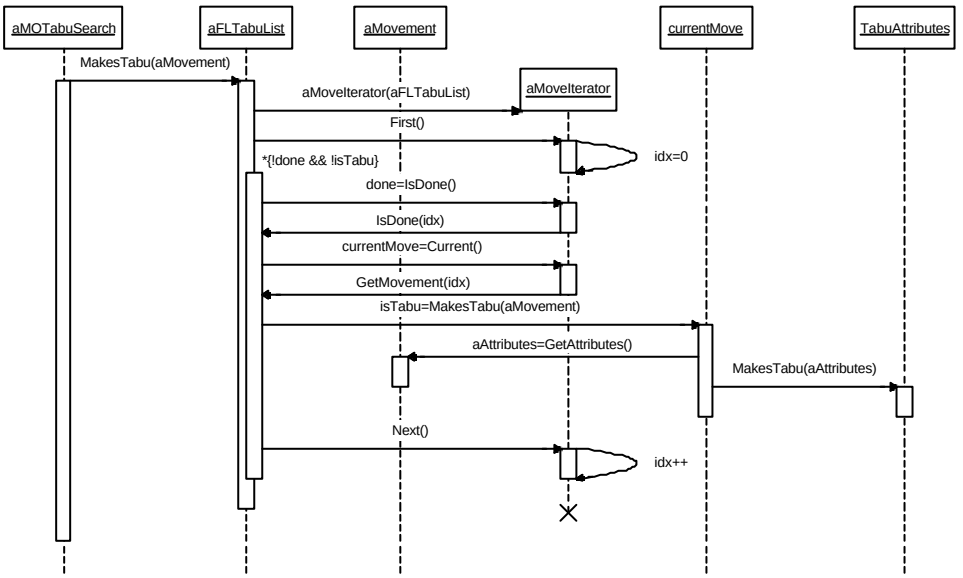


Figura 5.14 Diagrama de sequência para verificação do estado tabu

5.8 Extensão para vizinhanças variáveis

Esta extensão resulta da flexibilização da parte do *framework* relativa às vizinhanças, no sentido de considerar a possibilidade de dinamicamente, e de acordo com um determinado critério, se alterar a vizinhança. Para tal recorre-se à utilização do *padrão Estado*, numa aplicação

baseada em [Andreatta et al. 1998]. Este *padrão* permite a uma classe alterar o seu comportamento quando o seu estado interno se altera, simulando uma mudança de classe. Desta forma consegue-se uma explicitação das mudanças de estado, que corresponderá, no caso concreto das vizinhanças variáveis, ao gerador de movimentos.

Uma vez que a extensão é realizada por derivação, será necessário garantir a aplicação de um outro *padrão*, definido ao nível da classe base: o *padrão* Protótipo, que permite a criação de novas vizinhanças, para as diversas soluções da população inicial.

Estrutura e participantes

O diagrama de classes apresentado na Figura 5.15 apresenta a estrutura de classes para as vizinhanças variáveis.

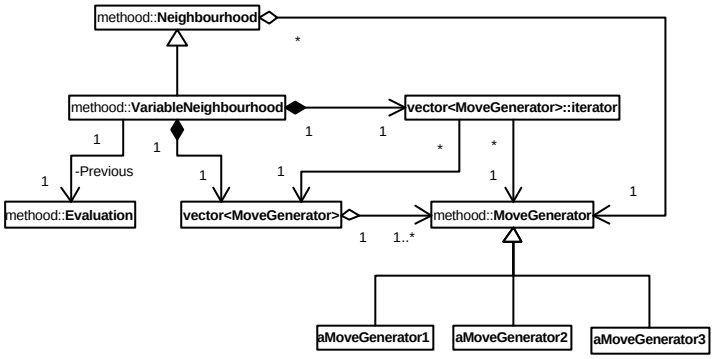


Figura 5.15 Diagrama de classes para vizinhanças variáveis

Os participantes nesta parte do *framework* são:

- *Vizinhança* (*Neighbourhood*).
- *Vizinhança variável* (*VariableNeighbourhood*). Classe derivada da vizinhança. Implementa a mudança de gerador de movimentos, de acordo com as regras das vizinhanças variáveis. Possui um vector de geradores de movimentos (*vector<MoveGenerator>*), sobre o qual itera com o respectivo iterador (*vector<MoveGenerator>::iterator*).
- *Avaliação anterior* (*Evaluation*$_{Previous}$). Classe de avaliação utilizada para salvaguardar a avaliação da solução anterior, por forma a permitir a sua comparação com a avaliação de uma nova solução base da vizinhança.
- *Gerador de movimentos (MoveGenerator).*

- *Geradores de movimentos concretos* (*MoveGenerator1, ...*). Implementam os diversos geradores de movimentos correspondentes às vizinhanças a considerar dinamicamente.
- *Vector de geradores de movimentos* (*vector<MoveGenerator>*). Contém os geradores de movimentos, na ordem em que deverão ser aplicados.
- *Iterador de vector de geradores de movimentos* (*vector<MoveGenerator>::iterator*). Iterador para acesso aos elementos do vector de geradores de movimentos.

Colaborações

Ao nível da aplicação do *padrão* Protótipo, sempre que tal seja solicitado, são criadas novas vizinhanças variáveis, que ficam associadas aos mesmos geradores de movimentos.

As restantes colaborações (Figura 5.16) têm lugar em dois cenários:

- *Alteração da solução actual* (*SET SOLUTION*). Quando se efectua uma alteração da solução actual, é realizada uma comparação dos produtos escalares dos pesos equalizados da nova solução pelos valores das funções objectivo para a avaliação anterior e para a nova solução. Se esta apresentar um melhor valor, a vizinhança regressa ao primeiro gerador.
- *Reinício da exploração da vizinhança* (*REFRESH*). Quando se reinicia a exploração da vizinhança, o gerador de movimentos em vigor passa a ser o gerador seguinte no vector de geradores. Constituem excepções as seguintes situações: se a vizinhança tiver regressado ao primeiro gerador numa situação de alteração da solução actual, será esse primeiro gerador que deverá explorar; no caso de a vizinhança ter atingido o último gerador, deverá regressar ao primeiro.

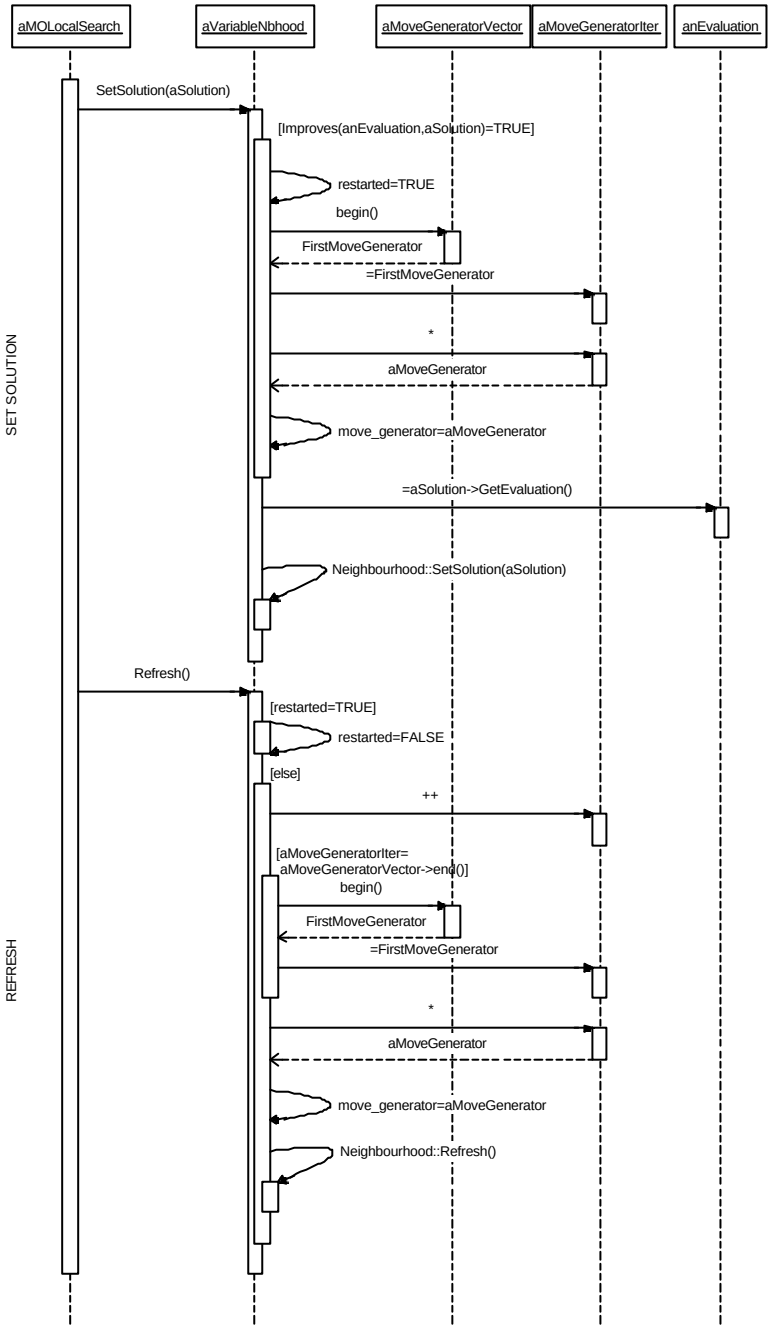


Figura 5.16 Diagrama de sequência para vizinhanças variáveis

5.9 Extensão para estratégias de listas de candidatos

As estratégias de listas de candidatos são resultado de uma outra extensão ao núcleo do *framework*, igualmente incidindo sobre a vizinhança. A perspectiva adoptada para esta extensão foi considerar as listas de candidatos como uma memória de movimentos potencialmente interessantes, que a vizinhança vai mantendo. Esta memória pode ser vista como uma extensão da própria vizinhança.

Ao pesquisar uma vizinhança, vão sendo colocados na lista de candidatos, os melhores movimentos de entre os já aí presentes e os novos movimentos pesquisados. A selecção do melhor movimento é feita a partir dos movimentos contidos na lista. Com a alteração da solução base da vizinhança, todos os movimentos são reavaliados e os movimentos não admissíveis são retirados.

Esta extensão estrutura-se sobre a aplicação de três *padrões de desenho*: o *padrão* Iterador, para acesso aos movimentos da lista de candidatos, o *padrão* Estratégia, na definição de uma família de estratégias de listas de candidatos, e o *padrão* Protótipo, herdando a utilização prescrita para a classe base vizinhança.

Estrutura e participantes

O diagrama de classes apresentado na Figura 5.17 apresenta a estrutura de classes para as estratégias de listas de candidatos.

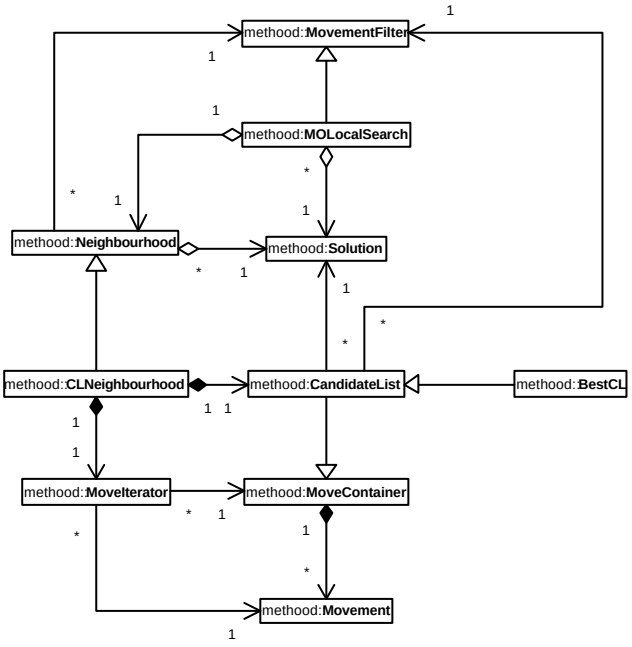


Figura 5.17 Diagrama de classes para estratégias de listas de candidatos

Os participantes nesta parte do *framework* são:

- *Vizinhança* (*Neighbourhood*).
- *Vizinhança com lista de candidatos* (*CLNeighbourhood*). Classe derivada da vizinhança. Implementa uma lista de candidatos (*CandidateList*).
- *Contentor de movimentos* (*MoveContainer*).
- *Iterador de movimentos* (*MoveIterator*).
- *Movimento* (*Movement*).
- *Solução* (*Solution*).
- *Algoritmo de pesquisa local multiobjectivo* (*MOLocalSearch*).
- *Filtro de movimentos* (*MovementFilter*).
- *Estratégia elite* (*CLBest*). Lista com comprimento fixo máximo na qual são mantidos os melhores movimentos. Para a inserção na lista, quando esta se encontra completamente preenchida, é realizada uma comparação dos produtos escalares dos pesos equalizados da solução actual pelos valores das funções objectivo para os movimentos presentes na lista e para o novo movimento. No caso de algum movimento presente na lista apresentar um valor pior, o novo movimento substituí-lo-á.

Colaborações

É possível identificar duas áreas principais de colaboração:

- *Repercussão sobre a lista de candidatos da interacção entre o algoritmo de pesquisa local e a vizinhança* (Figura 5.18).

  Nesta área encontra-se patente a aplicação do *padrão* Estratégia, com a vizinhança a encaminhar para a lista de candidatos, vários dos pedidos realizados pelo algoritmo de pesquisa local multiobjectivo.

  Os principais cenários presentes nesta área são:

  – *Alteração da solução actual* (*SET SOLUTION*). A alteração da solução actual é propagada à lista de candidatos.

  – *Reinício da exploração da vizinhança* (*REFRESH*). Quando se reinicia a exploração da vizinhança, a lista de candidatos é actualizada, da forma descrita adiante.

– *Verificação da elegibilidade de um movimento* (*SELECTABLE*). Se um movimento for aceitável pelos critérios do princípio meta-heurístico e for inserido na lista de candidatos, então será considerado elegível para execução sobre a solução actual. Nos restantes casos, não o será.

– *Selecção de um movimento* (*SELECTION*). Seleccionar-se-á o movimento com melhor valor do produto escalar, entre os respectivos valores dos critérios e os pesos equalizados da solução actual.

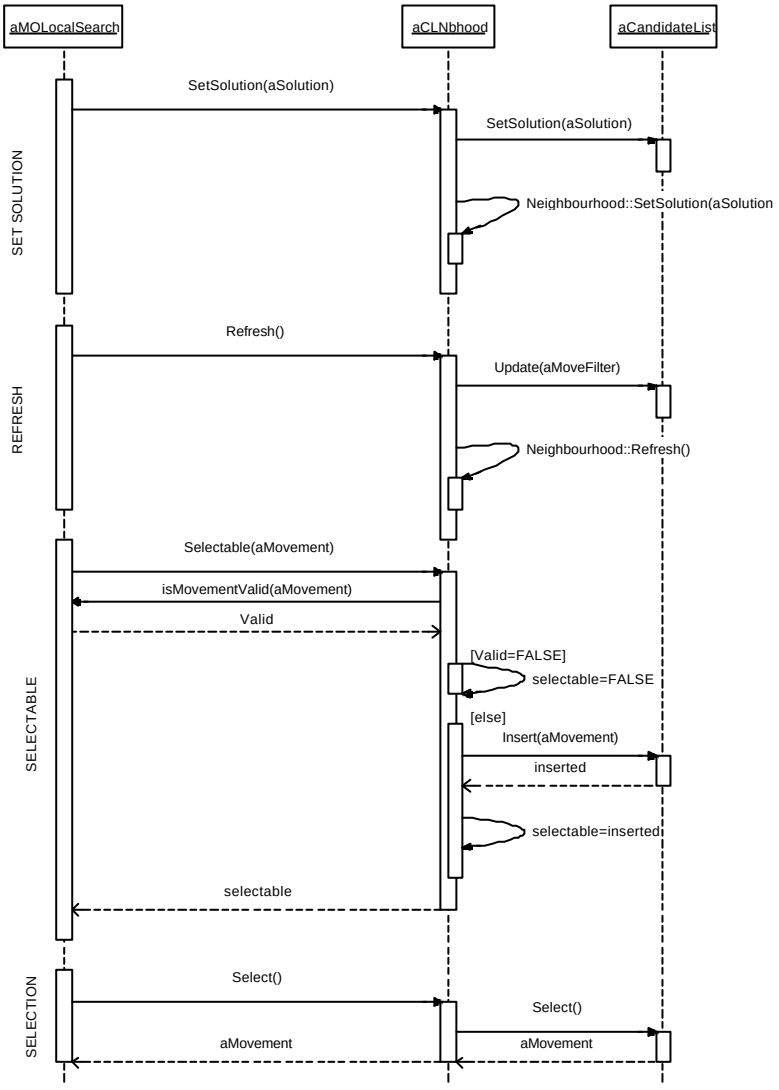


Figura 5.18 Diagrama de sequência para estratégias de listas de candidatos

- *Actualização dos movimentos* (Figura 5.19). São recalculadas as avaliações dos movimentos presentes na lista, e são retirados desta todos os movimentos não admissíveis pela nova solução ou pelo princípio meta-heurístico. O *padrão* Iterador é aplicado para a travessia e acesso aos movimentos da lista de candidatos.

Tal como para as vizinhanças variáveis, também neste caso a aplicação do *padrão* Protótipo envolve apenas a criação de vizinhanças com o mesmo tipo de lista de candidatos, sempre que tal seja solicitado.

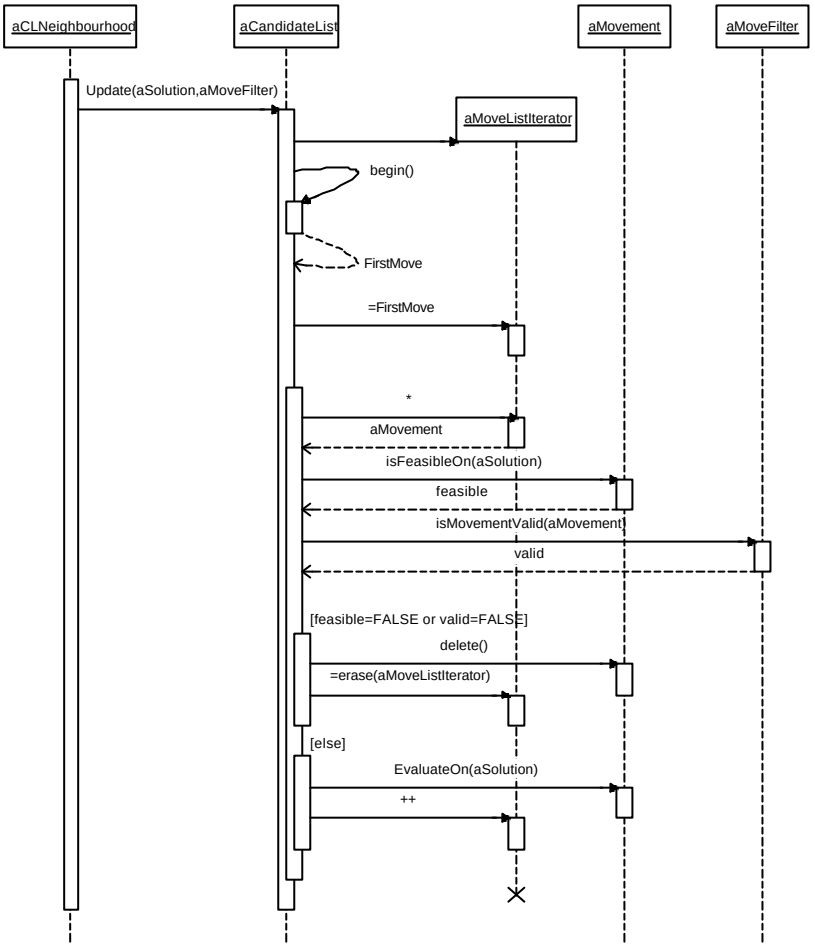


Figura 5.19 Diagrama de sequência para a actualização dos movimentos em estratégias de listas de candidatos

5.10 Extensão para paralelização e hibridização

A abordagem de paralelização e hibridização implementada no âmbito deste *framework* segue a sugestão apresentada no Capítulo 2, envolvendo vários algoritmos de pesquisa local que trabalham sobre sub-populações próprias, mas partilhando uma população global comum para a definição de pesos, a aproximação ao conjunto de soluções eficientes, as gamas e os factores de equalização de gamas. Esta abordagem resulta, em grande parte, de um esforço no sentido de identificar e isolar factores comuns aos diversos algoritmos de pesquisa local multiobjectivo. Neste sentido, a aplicação de alguns *padrões de desenho* foi determinante para viabilizar a sua implementação: a aplicação do *padrão* Estratégia permitiu consagrar um interface comum a todos os algoritmos de pesquisa local multiobjectivo, e a aplicação do *padrão* Método *Template* permitiu destacar os aspectos comuns das suas estruturas internas.

A abordagem considerada é uma hibridização de alto nível, pelo que, coerentemente, deverá ser localizada no *framework* a um nível superior ao dos algoritmos de pesquisa local. A opção tomada foi no sentido da localização ao nível dos *solvers* meta-heurísticos, que realizavam inicialmente a articulação de algoritmos construtivos e algoritmos de pesquisa local. A descrição detalhada dos *solvers* meta-heurísticos pode ser encontrada no Anexo B.

A criação desta extensão envolveu, no entanto, alterações ao nível de algumas classes mais básicas, para dotar os componentes comuns de regras de acesso. Esses componentes comuns são a aproximação ao conjunto de soluções eficientes, os factores de equalização de gamas, as gamas e as soluções. As classes referidas foram dotadas de um método *lock* e um método *unlock*. Para acesso à classe deve ser invocado o método *lock*, que assegura a exclusividade do acesso, logo que a classe esteja disponível. Quando o acesso à classe deixa de ser necessário, deve ser invocado o método *unlock*. Todos os acessos a estes componentes foram alterados para passar a utilizar este mecanismo. A sua utilização terá sempre algum impacto na eficiência dos algoritmos, de forma que são disponibilizadas duas versões para compilação do *framework*: com *multithread* ou sem *multithread*.

Na implementação particular realizada, os métodos empregam mecanismos do tipo *mutex* [Microsoft 1998], disponibilizados ao nível do sistema operativo. O *mutex* é um mecanismo de sincronização que permite a *threads* o acesso mutuamente exclusivo a um recurso, implementando os métodos referidos anteriormente, *lock* e *unlock*, para, respectivamente, obter ou libertar um determinado acesso.

Estrutura e participantes

O diagrama de classes apresentado na Figura 5.20 apresenta a estrutura de classes para a abordagem de hibridização e paralelização.

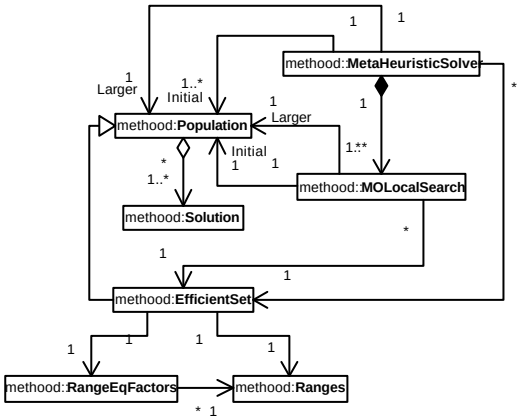


Figura 5.20 Diagrama de classes para a abordagem de hibridização e paralelização

Os participantes nesta parte do *framework* são:

- *Solver meta-heurístico* (*MetaHeuristicSolver*). Ao nível de paralelização e hibridização, é responsável pelo lançamento de *threads* paralelos com vários algoritmos de pesquisa local multiobjectivo (*MOLocalSearch*). Cada um destes algoritmos terá a sua própria população inicial ($Population_{Initial}$) de soluções (*Solution*), sendo a população alargada ($Population_{Larger}$) formada pelo conjunto de todas as populações iniciais. Os serviços dos algoritmos de pesquisa local multiobjectivo (*MOLocalSearch*) permitirão determinar uma aproximação ao conjunto de soluções eficientes (*EfficientSet*), que será comum aos vários algoritmos, bem como as gamas (*Ranges*) e os factores de equalização de gamas (*RangeEqFactors*), que dependem daquele conjunto.
- *População* (*Population*).
- *Solução* (*Solution*).
- *Algoritmo de pesquisa local multiobjectivo* (*MOLocalSearch*).
- *Aproximação ao conjunto de soluções eficientes (EfficientSet).*
- *Factores de equalização de gamas* (*RangeEqFactors*).
- *Gamas* (*Ranges*).

Colaborações

Os principais cenários de colaboração a destacar, no âmbito desta abordagem de hibridização e paralelização, são a sincronização dos acessos aos componentes comuns e o lançamento dos vários *threads* pelo *solver* meta-heurístico.

O primeiro cenário, com a regulação do acesso através do uso dos métodos *lock* e *unlock*, foi já sucintamente descrito no início desta subsecção. No segundo cenário (Figura 5.21), após a construção das populações iniciais e da população alargada, e a configuração dos algoritmos de pesquisa local, são sucessivamente lançados os vários *threads* de pesquisa local, e aguarda-se, também sucessivamente, o término da respectiva execução.

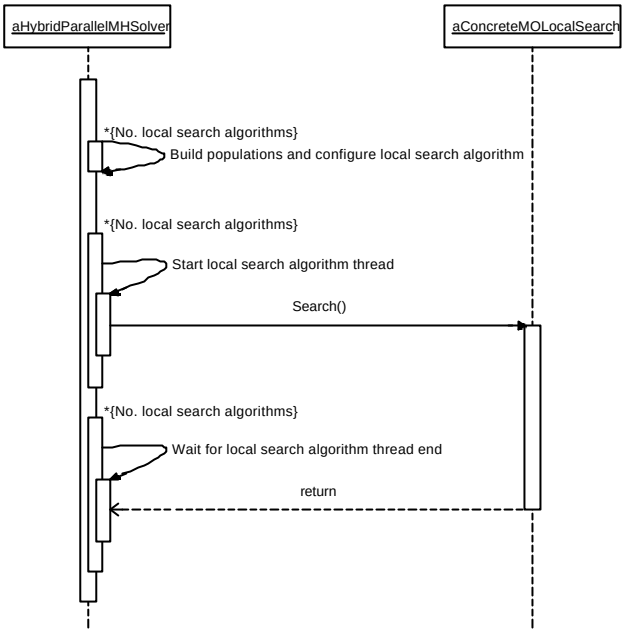


Figura 5.21 Diagrama de sequência para o lançamento de *threads* na abordagem de hibridização e paralelização

5.11 Relações do METHOOD com outras abordagens

Apresentam-se de seguida, e de uma forma sintética, as principais orientações que estiveram na base do desenvolvimento do METHOOD, salientando-se as diferenças mais importantes relativamente a outras abordagens descritas na literatura, e conferindo especial destaque aos aspectos considerados inovadores.

A principal motivação para o desenvolvimento deste *framework* encontra-se presente na área das abordagens OO para meta-heurísticas, desde o trabalho pioneiro de [Ferland et al. 1996]: tirar partido do paradigma OO para beneficiar a aplicação e a comparação de meta-heurísticas. A aplicação do trabalho desenvolvido por esses autores está, no entanto, restrita a problemas de um tipo específico. O mesmo sucede com o *framework TabooBuilder* [Graccho, Porto 1999] que, adicionalmente, se restringe também a abordagens baseadas em TS. No caso do METHOOD, bem como de outras abordagens encontradas na literatura, a opção é no sentido da generalização do tipo de problemas que se podem considerar e da disponibilização de vários algoritmos, desenvolvidos a partir de infra-estruturas comuns.

É do *framework Searcher* [Andreatta et al. 1998] que resultaram as influências mais significativas sobre o presente trabalho, em particular ao nível da utilização de *padrões de desenho*, da preocupação com aspectos de flexibilidade e do suporte explícito para algoritmos construtivos. O facto de se lidar no presente trabalho com meta-heurísticas multiobjectivo conduz às principais diferenças em relação ao trabalho de [Andreatta et al. 1998], com a

introdução de componentes relativos à avaliação multiobjectivo, ao tratamento de populações de soluções, etc. Uma outra diferença fundamental reside na separação entre os aspectos de representação e de semântica das soluções. Aliás, já em [Andreatta et al. 1998] se referia esta questão, relativamente à diferenciação da abordagem de [Woodruff 1997]. Por exemplo, enquanto no *Searcher*, um pedido de execução de um movimento sobre uma solução é dirigido a essa solução, no METHOOD o pedido é dirigido ao próprio movimento. Desta forma, a solução fica isolada de diferentes semânticas particulares, o que simplificará a sua reutilização noutros contextos, como será o caso, por exemplo, da integração no METHOOD, de meta-heurísticas baseadas em recombinação.

Um aspecto que não tem merecido tratamento explícito na maioria das abordagens da literatura é a disponibilização de abordagens de hibridização e paralelização. Em [Woodruff 1997] é referida, embora sem uma apresentação específica, a consideração de estratégias de hibridização. No trabalho de [Schaerf et al. 1999], são apresentadas algumas estratégias muito simples. No METHOOD este aspecto é tratado explicitamente, propondo-se uma abordagem que permite realizar paralelização do tipo *"threads"* multipesquisa e cooperativa, e hibridização de alto nível, co-evolutiva.

Fora do domínio das abordagens OO, deverá ser destacado o *template* de [Vaessens et al. 1995], que constitui a principal referência para a aplicação do *padrão* Método *Template* neste trabalho.

Inúmeras diferenças de pormenor poderiam ser aqui apresentadas, em particular ao nível das opções de desenho. Será, no entanto, de maior importância destacar os principais aspectos de inovação do METHOOD, que constituem factores fundamentais de diferenciação em relação a outras abordagens:

- a utilização explícita do paradigma OO para suportar flexibilização em meta-heurísticas, designadamente estratégias de listas de candidatos, vizinhanças variáveis, paralelização e hibridização;
- a aplicação à área das meta-heurísticas multiobjectivo, com destaque para a construção de um *template* de pesquisa local multiobjectivo, a adaptação a esta área de estratégias de listas de candidatos e vizinhanças variáveis, e a criação de uma abordagem de hibridização e paralelização de alto nível.

Para concluir, ao nível da evolução futura do *framework*, dois aspectos abordados na literatura poderão vir a assumir particular relevância:

- O *framework* apresentado em [Fink et al. 1998b] baseia-se na utilização de polimorfismo estático, alternativa que, segundo os autores referidos, será potencialmente mais eficiente do que a utilização de mecanismos de herança.

Nesse sentido, a evolução do METHOOD irá, portanto, passar, muito provavelmente, pela adopção de polimorfismo estático.

- A linguagem *Localizer* [Michel, van Hentenryck 1998] sugere uma extensão ao *framework*, para a qual este estará potencialmente preparado, no sentido de permitir uma configurabilidade de alto nível das abordagens meta-heurísticas disponibilizadas.

5.12 Conclusões

A convicção de que uma abordagem OO potencia a disponibilização de uma arquitectura de base flexível para a construção, aplicação e comparação de meta-heurísticas multiobjectivo, integrando estratégias de flexibilização genéricas, em contextos de desenvolvimento de aplicações práticas e investigação na área da Optimização Combinatória Multiobjectivo, constituiu a principal motivação para o desenvolvimento do *framework* METHOOD (*METaHeuristics Object-Oriented Development*).

A descrição de um *framework* incide no seu objectivo, no seu desenho detalhado e nas instruções para a sua utilização. Os primeiros dois aspectos são objecto de atenção neste capítulo, enquanto o terceiro é tratado no Capítulo 6. A parte mais substancial da descrição corresponde ao desenho detalhado que inclui, para cada parte do *framework* considerada, as estruturas de classes e respectivas colaborações. Apenas as partes do *framework* relacionadas com a pesquisa local multiobjectivo foram, por motivos de espaço e organização do texto, tratadas no âmbito deste capítulo. As restantes partes são descritas no Anexo B.

Um elemento fundamental para o desenho do *framework* é a utilização de *padrões de desenho*. Do "catálogo" de [Gamma et al. 1995], foram utilizados os seguintes *padrões*: Estratégia (*"Strategy"*), Método *Template* (*"Template Method"*), Iterador (*"Iterator"*), Protótipo (*"Prototype"*) e Estado (*"State"*).

Ao nível da avaliação de soluções e movimentos num contexto multiobjectivo, o *framework* disponibiliza um vector de critérios, associado a um conjunto de operações mais frequentes, e a um vector de pesos e um vector pesos equalizados. A cada critério está associada uma função de avaliação, distinta para soluções e movimentos.

A pesquisa local multiobjectivo assenta essencialmente na aplicação do *padrão* Método *Template*, que implementa o *template* de pesquisa local multiobjectivo, definido em 3.3.1. É a concretização deste *template* em classes derivadas que permite a extensão para duas meta-heurísticas concretas baseadas em pesquisa local multiobjectivo: PSA e MOTS*.

A flexibilização da parte do *framework* relativa às vizinhanças permite a consideração de vizinhanças variáveis e de estratégias de listas de candidatos, e a sua integração com o

template de pesquisa local multiobjectivo, de acordo com as propostas apresentadas no Capítulo 3. O esforço realizado no sentido de identificar e isolar factores comuns aos diversos algoritmos de pesquisa local multiobjectivo foi determinante para a concepção da abordagem de hibridização e paralelização de alto nível, igualmente apresentada no Capítulo 3, e na qual toda a parte do *framework* relativa a algoritmos básicos é reutilizada.

Relativamente a outras abordagens presentes na literatura, o METHOOD diferencia-se essencialmente pela utilização explícita do paradigma OO para suportar flexibilização em meta-heurísticas, e pela sua aplicação à área das meta-heurísticas multiobjectivo. A procura de uma melhoria da eficiência do *framework* e o desenvolvimento de uma extensão para configuração de alto nível das abordagens meta-heurísticas disponibilizadas, são trajectórias de evolução futura do *framework* que vão ao encontro de algumas tendências encontradas na literatura.

No desenvolvimento do *framework* foi possível, desde logo, constatar o nível efectivamente elevado de flexibilidade e configurabilidade conferido pelas características básicas do paradigma OO. Em grande parte do *framework*, tal é indissociável da aplicação de *padrões de desenho*. Com efeito, sendo concebidos como soluções genéricas para problemas de desenho recorrentes, os *padrões de desenho* apresentam, logo à partida, e com base nas características básicas do paradigma OO, elevados níveis de flexibilidade e configurabilidade.

A modelação efectiva do domínio e a separação de facetas permitem estabelecer uma infra-estrutura conceptualmente clara, o que constitui um apoio indispensável à concepção e implementação de algoritmos de grande dimensão e complexidade, como é o caso das meta-heurísticas multiobjectivo. Por outro lado, estes aspectos incentivam efectivamente a construção de novas meta-heurísticas ou variantes, através de extensões por composição ou derivação de subclasses, conforme sucede no *framework*, com a implementação e integração de um conjunto de estratégias genéricas.

É ainda de salientar o elevado potencial de reutilização atingido, constatável no desenho de diversas partes do *framework*, bem como no esforço de desenvolvimento envolvido, que é significativamente reduzido. A disponibilização de um conjunto relativamente alargado de técnicas algorítmicas e variantes, para contextos multiobjectivo, é realizada, no total, por cerca de 50 classes, inteiramente reutilizáveis, e exigiu um esforço de desenvolvimento de cerca de 3 pessoas-mês.

6
CASO DE ESTUDO: ESCALONAMENTO MULTIOBJECTIVO DE TAREFAS NUMA MÁQUINA

Neste capítulo é apresentado o problema abordado no caso de estudo - escalonamento multiobjectivo de tarefas numa máquina, minimizando o atraso máximo, o número de tarefas atrasadas e a soma ponderada dos atrasos - e a aplicação a esse problema do *framework* desenvolvido.

Esta aplicação permite completar a descrição do *framework*, bem como verificar o seu potencial, em particular, na aplicação directa a problemas de Optimização Combinatória Multiobjectivo. O caso de estudo complementa assim outros resultados como os associados à criação de extensões ao nível dos componentes elementares e da composição de algoritmos (apresentados no Capítulo 5), ou à comparação de diversas configurações de algoritmos (descritos no Capítulo 7).

Na secção 6.1 introduzem-se os conceitos básicos de escalonamento, que são na secção 6.2 desenvolvidos para o caso particular de uma máquina. Na secção 6.3 apresenta-se uma revisão das abordagens de pesquisa local aplicada ao escalonamento numa máquina, encontradas na literatura. Na secção 6.4 estende-se esta revisão ao escalonamento multiobjectivo numa máquina e apresenta-se o problema em consideração. A aplicação do *framework* é apresentada na secção 65, incidindo sobre os dados do problema, soluções, avaliação, algoritmos construtivos e vizinhanças. Um pequeno conjunto de conclusões encerra o capítulo.

6.1 Escalonamento

6.1.1 Contexto e definição

Os problemas de *escalonamento* (*"scheduling"*) têm sido um constante objecto de estudo no âmbito da optimização, como resultado natural da sua enorme importância prática. Esta importância permanece actual, conforme revela o elevado número de publicações na área que continuamente vão surgindo, apresentando novas aplicações e evoluções.

É precisamente de um livro recente [Pinedo, Chao 1999] que se transcreve uma descrição sucinta dos principais aspectos envolvidos no *escalonamento*:

> «O *escalonamento* é um processo de tomada de decisão que assume um papel importante na maioria das empresas de serviços ou industriais. É utilizado em *procurement* e produção, em transporte e distribuição, e no processamento e transmissão de informação. A função de *escalonamento*, numa empresa, usa técnicas matemáticas ou métodos heurísticos para afectar recursos limitados ao processamento de tarefas. Uma afectação adequada dos recursos permite à empresa optimizar os seus objectivos e atingir as suas metas. Os *recursos* podem ser máquinas numa instalação fabril, passagens num aeroporto, equipas num local de construção, ou unidades de processamento num ambiente computacional. As *tarefas* podem ser operações numa instalação fabril, partidas e chegadas num aeroporto, andares num projecto de construção, ou programas de computador a executar. Cada tarefa pode ter um nível de prioridade, uma data de arranque mais cedo e uma data de entrega. Os objectivos podem também assumir variadas formas, como minimizar o tempo necessário para completar todas as tarefas ou minimizar o número de tarefas terminadas após as respectivas *due dates*.»

A área de aplicação em que se enquadra o trabalho apresentado nesta dissertação é também a área em que o tratamento destes problemas surge provavelmente com maior frequência: *escalonamento de máquinas* (*"machine scheduling"*). Na terminologia da área, um *recurso* é tipicamente designado *máquina*, e uma entidade a ser processada numa máquina, *tarefa* (*"job"*).

Nesta área mais restrita, além do recente e já referido [Pinedo, Chao 1999], constituem referências fundamentais [Baker 1974], [French 1982], [Blazewicz et al. 1986] e [Blazewicz et al. 1993].

6.1.2 Caracterização de tarefas e notação

Os elementos que, de forma geral, caracterizam uma tarefa são apresentados, conjuntamente com a respectiva notação, na Tabela 6.1.

Característica	Notação	Descrição
Tempo de processamento	p_{ij}	Duração do processamento da tarefa *j* na máquina *i*.
Data de lançamento (*"release date"*)	r_j	Data em que a tarefa *j* chega ao sistema, ou seja, a data mais cedo, a partir da qual pode ser processada.
Data de entrega (*"due date"*)	d_j	Data de envio ou entrega ao "cliente", associada à tarefa *j*.
Peso	w_j	Factor de prioridade, denotando a importância da tarefa *j* em relação a outras tarefas do sistema. Pode representar o custo de manter a tarefa no sistema, um custo de posse ou stock, ou o valor já acrescentado à tarefa.

Tabela 6.1 Características das tarefas e respectiva notação

Como resultado do processo de escalonamento, fica definido o instante em que a tarefa *j* é terminada na máquina *i*. Esse instante é designado por C_{ij}. C_j representa o instante em que a tarefa *j* abandona o sistema.

6.1.3 Características e restrições no processamento de tarefas

Apesar de constituir uma área mais restrita dentro do escalonamento, o escalonamento de máquinas compreende uma grande variedade de situações, em particular relativamente às características e restrições específicas em que pode ocorrer o processamento de tarefas. Em [Pinedo, Chao 1999] são enumeradas algumas das características e restrições mais frequentes:

- Restrições de precedência, que se traduzem em uma tarefa só poder arrancar após o término de um conjunto de outras tarefas.
- Restrições às rotas (sequências de máquinas) que uma tarefa deve seguir através de um sistema.
- Restrições de movimentação de materiais, devidas às características dos sistemas que movimentam as "tarefas" entre centros de trabalho. Por exemplo, a existência de um espaço de *buffer* limitado impõe um limite ao nível de trabalho em curso.

- Custos e tempos de *setup* dependentes da sequência. A reconfiguração ou limpeza das máquinas entre tarefas frequentemente depende da tarefa que acabou de ser realizada e da tarefa que vai ser realizada de seguida.
- "Preempção". Consiste na possibilidade de interrupção da realização de uma tarefa para a realização de outras (por exemplo, de maior prioridade).
- Restrições de tempo de espera e espaço de armazenamento.
- Produção para *stock* ou por encomenda. A produção para *stock* tipicamente não implica considerar datas de entrega rígidas.
- Restrições de selecção de máquinas. Uma tarefa tem que ser realizada numa máquina de entre um conjunto restrito de máquinas.
- Restrições relacionadas com ferramentas ou recursos.
- Restrições de escalonamento de pessoal (associado à operação das máquinas).
- *Deadlines* (datas em que uma tarefa forçosamente terá que estar concluída) e *time-lags* (intervalos de tempo a garantir entre realizações de tarefas) positivos e negativos.

6.1.4 Configurações de máquinas

Um outro elemento determinante para a variedade de situações a considerar é, naturalmente, a configuração das máquinas. De entre as mais importantes, destacam-se, com base em [Pinedo, Chao 1999] e [Madureira 1995], as mais básicas:

- **Modelos de uma máquina (*"single-machine"*)**

  Os modelos de uma máquina são de aplicação importante em situações de estrangulamento da produção ou em abordagens de decomposição de problemas complexos. Têm sido alvo de um estudo exaustivo, com grande variedade de condições, restrições e objectivos. Existe um conjunto de regras, muitas das quais de identificação e aplicação simples, que produzem frequentemente soluções óptimas para estes modelos.

- **Modelos de máquinas paralelas**

  Estes modelos consistem na generalização dos modelos de uma máquina, permitindo que qualquer tarefa possa ser realizada numa de um conjunto de máquinas "funcionalmente" idênticas. As suas variantes estão relacionadas com as máquinas serem ou não idênticas. São de aplicação importante também em situações de estrangulamento.

Considera-se ainda, neste modelo, que cada tarefa é constituída por uma única operação. Os três modelos que se apresentam de seguida designam-se por "multi-operação" uma vez que, ao contrário dos anteriores, cada tarefa é constituída por um conjunto de operações.

- **Modelos de *flow-shop***

  Nesta configuração todas as tarefas se decompõem na mesma sequência de operações a executar em diversas máquinas distintas, ou seja, as tarefas visitam todas as máquinas na mesma ordem. A generalização com máquinas paralelas em cada estádio é designada *flexible flow shop*.

- **Modelos de *open-shop***

  Ocorrem quando as tarefas se decompõem em conjuntos de operações a executar em diversas máquinas distintas, sem requisitos de sequência (situação pouco frequente, na prática). A generalização com máquinas paralelas em centros de trabalho é designada *flexible open shop*.

- **Modelos de *job-shop***

  Nos modelos de *job-shop* as tarefas decompõem-se em sequências de operações não necessariamente idênticas, a executar em diversas máquinas distintas. Nestes ambientes as tarefas não visitam as máquinas na mesma ordem. A generalização com máquinas paralelas em centros de trabalho é designada *flexible job shop*.

6.1.5 Objectivos

Existe uma grande variedade de objectivos que tipicamente são considerados em actividades de escalonamento. Em quase todas as situações práticas, seria fortemente desejável ter em consideração, simultaneamente, mais do que um destes objectivos. Mas apesar disso, a grande maioria das abordagens da literatura só considera um objectivo, sendo muito recente o aparecimento regular de trabalhos tratando o carácter multicritério dos problemas de escalonamento. Em [Pinedo, Chao 1999] são destacados os objectivos encontrados mais regularmente:

- *throughput*, que consiste na taxa de *output* de uma instalação;
- *makespan*, que consiste no tempo que medeia entre o início de operação e o instante em que a última tarefa abandona o sistema;

- objectivos relacionados com datas de entrega:
  - atraso máximo;
  - número de tarefas atrasadas;
  - atraso total ou médio;
  - atraso ponderado, baseado na multiplicação do atraso de cada tarefa pelo correspondente peso unitário;
- custos de *setup*;
- custos de *stock* de produto em curso de fabrico;
- custos de *stock* de produto acabado;
- custos de pessoal.

6.1.6 Classificação dos problemas

Os pressupostos assumidos na literatura para a maioria dos problemas de escalonamento (e que aqui vão também ser adoptados) são o carácter determinístico dos dados, a independência dos tempos de *setup* em relação à sequência de processamento, a imediata disponibilidade de todas as tarefas (r_j=0), a inexistência de precedências entre tarefas e a não possibilidade de interrupção das tarefas em processamento (preempção não permitida).

Em [Graham et al. 1979] é sugerido um esquema de descrição e identificação dos diferentes tipos de problemas de escalonamento.

De acordo com este esquema, os problemas são denotados por $\alpha \mid \beta \mid \gamma$, em que:

- α é o tipo de problema, de entre as cinco configurações de máquinas apresentadas anteriormente.
- β são os desvios em relação aos pressupostos normais, usando a notação anteriormente apresentada, ***prec*** para a indicação de ocorrência de precedências e ***pntm*** para a preempção, ou r_j para a existência de datas de lançamento diferentes.
- γ representa a função objectivo. Na apresentação dos problemas de escalonamento numa máquina será introduzida a notação respeitante a este campo. A título de exemplo apresentam-se os três objectivos considerados no problema multiobjectivo a estudar: atraso máximo - L_{max}; número de tarefas atrasadas - SU_j (U_j = 0 se a tarefa j não está atrasada, 1 se está atrasada); soma ponderada dos atrasos - $\Sigma w_j T_j$.

Note-se que, posteriormente, foram propostas várias extensões desta classificação, aqui não consideradas, por envolverem aspectos não relevantes neste trabalho.

6.1.7 Regras de prioridade básicas

As regras de prioridade (*"dispatching rules"*) permitem estabelecer prioridades entre as tarefas que aguardam processamento numa máquina, de acordo com um esquema que, de forma geral, pode tomar em consideração os atributos das tarefas e das máquinas, bem como o estado do sistema no instante actual. O comportamento destas regras consiste em, sempre que uma máquina ficar disponível, inspeccionar as tarefas a aguardar processamento e seleccionar a que tiver maior prioridade.

Existem inúmeras regras básicas, entre as quais em [Pinedo, Chao 1999] são destacadas as apresentadas na Tabela 6.2.

Designação	**Abrev.**	**Descrição**	**Objectivo**
Service in random order	SIRO	Sempre que uma máquina é disponibilizada, a tarefa seguinte é seleccionada aleatoriamente de entre as que aguardam processamento.	-
Earliest release date first	ERD	É conferida prioridade à tarefa com menor data de lançamento.	Minimiza a variação dos tempos de espera das tarefas.
Earliest due date first	EDD	Possui prioridade a tarefa com menor data de entrega.	Tende a minimizar o atraso máximo entre tarefas aguardando processamento. No caso de uma máquina, minimiza o atraso máximo.
Minimum slack first	MS	É atribuída prioridade à tarefa com menor folga ($\max(d_j - p_j - t, 0)$) no instante de disponibilização da máquina.	Tende a minimizar objectivos relacionados com datas de entrega.
Weighted shortest processing time first	*WSPT*	É conferida prioridade à tarefa com maior rácio w_j / p_j. A variante com pesos iguais é designada *shortest processing time first* (SPT).	Tende a minimizar a soma ponderada dos instantes de conclusão. No caso de uma máquina, essa soma é minimizada.
Longest processing time first	*LPT*	Possuem maior prioridade as tarefas com maior tempo de processamento.	Tende a balancear cargas em máquina paralelas.
Shortest setup time first	*SST*	Confere-se maior prioridade às tarefas com menores tempos de *setup.*	Tende a minimizar o *makespan* e maximizar o *throughput*.
Least flexible job first	*LFJ*	Aplicável em situações em que as tarefas podem ser realizadas num subconjunto de máquinas. A prioridade é atribuída à tarefa com menor número de alternativas, de entre todas as tarefas que possam ser realizadas no conjunto de máquinas disponíveis.	Tende a minimizar o *makespan* e maximizar o *throughput*.

Critical path	*CP*	Aplicável quando existem restrições de precedência. É conferida maior prioridade à primeira tarefa da maior sequência de tempos de processamento no grafo de restrições de precedência.	Tende a minimizar o *makespan*.
Largest number of successors	*LNS*	Aplicável quando existem restrições de precedência. É atribuída maior prioridade à tarefa que possuir o maior número de tarefas que a sucedem.	Tende a minimizar o *makespan*.
Shortest queue at next operation	*SQNO*	Sempre que uma máquina é disponibilizada, é seleccionada a tarefa para a qual a máquina seguinte da sua rota possui a menor fila.	Tende a minimizar a *machine idleness.*

Tabela 6.2 Regras de prioridade básicas

6.1.8 Regras de prioridade compostas

Situações com objectivos mais complexos, como combinações de vários objectivos, ou objectivos que são função do tempo ou do conjunto de tarefas que aguardam processamento, exigem regras de prioridade que considerem um maior número de parâmetros. Algumas destas regras são combinações das regras básicas apresentadas anteriormente e são designadas *regras compostas* [Pinedo, Chao 1999].

Uma regra composta é, em geral, uma expressão de ordenação que combina diversas regras básicas. A cada regra básica presente na regra composta corresponde um factor de escala, que determina a contribuição da regra na expressão de ordenação. Os factores de escala podem ser pré-determinados ou função do conjunto de tarefas a seleccionar. Neste último caso são, com frequência, função de um conjunto de estatísticas que caracterizam a instância em causa da forma mais precisa possível. O "mapeamento" destas estatísticas no factor de escala é, em geral, pré-determinado.

Um exemplo de regra composta é a heurística *apparent tardiness cost* (ATC) [Vepsalainen, Morton 1981]. Esta regra aplica-se ao problema de uma máquina com *n* tarefas (todas disponíveis no instante 0) e com o objectivo de *minimização do atraso ponderado*. Trata-se de um problema de difícil resolução (NP-difícil) e para o qual não são conhecidos algoritmos eficientes.

A heurística ATC combina as regras WSPT e MS, determinando, para cada tarefa a aguardar processamento, um índice de ordenação, que é função do instante *t* em que a máquina fica livre e dos atributos p_j, w_j e d_j das restantes tarefas. É seleccionada a tarefa com o índice mais elevado. O índice é definido como

$$I_j(t) = \frac{w_j}{p_j} \exp\left(- \frac{\max(d_j - p_j - t, 0)}{K\,\overline{p}} \right),$$

em que *K* é o parâmetro de escala (designado parâmetro de *"look-ahead"*), que pode ser determinado empiricamente, e $\overline{p}$ é a média dos tempos de processamento das restantes tarefas. Se *K* for muito elevado, a regra ATC reduz-se a WSPT. Se for muito reduzido e não existirem tarefas fora de prazo, a regra reduz-se a MS. Se *K* for muito reduzido e existirem tarefas fora de prazo, a regra reduz-se a WSPT aplicado às tarefas fora de prazo. Os autores referidos recomendam, em geral, um valor de 2 para *K*.

6.2 Escalonamento numa máquina

6.2.1 Descrição do problema básico

Um problema básico de escalonamento numa máquina (SMSP, da designação inglesa *"Single Machine Scheduling Problem"*) tem os seguintes pressupostos:

- existe um conjunto de tarefas (j=1, 2, ..., n) independentes, caracterizadas por tempos de processamento p_j e data de entrega d_j, e cujo processamento não pode ser interrompido;
- as tarefas são processadas numa única máquina, que pode processar apenas uma tarefa de cada vez e está continuamente disponível.

Assume-se um horizonte temporal discretizado, com tempo *t* inteiro. O período de tempo *t* começa no instante *t*-1 e termina no instante *t*, sendo considerados os períodos 1, 2, ..., *T*. Com este enquadramento, um calendário de execução das tarefas é completamente descrito por um conjunto de *n* tempos de início de execução t_j.

A aplicação de restrições ao processamento de tarefas dá origem a variantes sobre este caso base, considerando, por exemplo, tempos de *setup* dependentes da sequência, precedência, preempção ou datas de lançamento.

A bibliografia neste domínio é bastante extensa, o que se prenderá, em parte, com a conjugação entre a simplicidade de descrição do problema e o seu interesse, bem como com a abundância de variantes. Constituem referências gerais úteis, uma bibliografia anotada apresentada em [Hoogeveen et al. 1997] e recensões bastante extensas em [Gupta, Kyparisis 1987] e [Lawler et al. 1993]. Sínteses de resultados sobre a complexidade computacional das

diversas variantes são apresentadas, entre outros, em [Baker 1974, Lawler 1983], [Sousa 1989] e [Blazewicz et al. 1991].

6.2.2 Critérios

O problema, básico ou nas suas diversas variantes, pode ser apreciado segundo uma variedade de pontos de vista ou critérios. Para a definição desses critérios será conveniente, em primeiro lugar, definir alguns valores relevantes:

- Tempo de conclusão (C_j - *"completion time"*) - instante de tempo em que o processamento da tarefa *j* é concluído.

$$C_j = t_j + p_j$$

- Atraso (L_j - *"lateness"*) - diferença entre o tempo de conclusão e a data desejada de entrega. Os atrasos negativos são denominados *"earliness"* e os positivos *"tardiness"*.

$$L_j = C_j - d_j$$

- Atraso positivo (*"tardiness"*) - atraso na conclusão de processamento de uma tarefa.

$$T_j = \max\{0, L_j\}$$

- Atraso negativo (*"earliness"*) - adiantamento na conclusão de processamento de uma tarefa. Pode representar custos de *stock* de produto acabado, custos de deterioração e obsolescência ou custos de oportunidade devidos a investimento improdutivo.

$$E_j = \max\{0, -L_j\}$$

- Penalização unitária dos atrasos (U_j - *"unit penalty"*).

$$U_j(C_j) = \begin{cases} 0 \text{ se } C_j \le d_j \\ 1 \text{ se } C_j > d_j \end{cases}$$

Exemplo 6.1 (Elaborado a partir de um exemplo de [Madureira, Sousa 1996])

Considere-se a instância de problema de escalonamento de 5 tarefas numa máquina, cujos parâmetros são descritos na parte esquerda da Tabela 6.3. Na parte direita da mesma tabela, são apresentados, para a sequência [3, 1, 2, 5, 4], os valores dos tempos de conclusão, atrasos positivos, atrasos negativos e penalização unitária dos atrasos.

Tarefa *j*	p_j	d_j	w_j	C_j	L_j	T_j	E_j	U_j
1	2	5	1	3	-2	0	2	0
2	4	7	6	7	0	0	0	0
3	1	11	2	1	-10	0	10	0
4	3	9	3	13	6	6	0	1
5	3	8	2	10	2	2	0	1

Tabela 6.3 Instância de escalonamento de tarefas numa máquina e respectiva caracterização para a sequência específica [3, 1, 2, 5, 4]

Uma importante distinção ao nível dos critérios, divide-os em critérios *regulares* e *não-regulares*.

Um critério diz-se *regular* se for uma função monótona crescente com qualquer tempo de conclusão. Sendo a função $f(C_1, C_2, ..., C_n)$ o valor de um critério regular para uma determinada sequência de tarefas e $f(C'_1, C'_2, ..., C'_n)$ o valor de uma outra sequência para a mesma função, se $f(C_1, C_2, ..., C_n) < f(C'_1, C'_2, ..., C'_n)$, então $C_j < C'_j$ para pelo menos uma tarefa *j*.

Em problemas sem datas de lançamento nem *"deadlines"* e com critérios regulares, demonstra-se que apenas é necessário considerar como soluções calendários sem *"idle time"*. É, então, suficiente determinar uma permutação óptima das tarefas, pelo que o problema se torna um problema de *sequenciamento*.

São critérios regulares:

- Funções do tipo $\sum f_j = \sum f_j(C_j)$, como o número ponderado das tarefas em atraso $\sum w_j U_j$ ou a soma ponderada dos atrasos $\sum w_j T_j$.

- Funções do tipo $f_{\max} = \max_{j=1,2,\ldots,n} f_j(C_j)$, como o *makespan* $C_{\max}$ ou o atraso máximo $L_{\max}$.

Os critérios não-regulares não são monotonamente crescentes com os tempos de conclusão das tarefas. A maioria das propriedades válidas para problemas com funções objectivo regulares não se mantêm quando o objectivo é não-regular, dificultando a pesquisa de escalonamentos óptimos.

De entre as funções objectivo que se enquadram nesta classificação, apresentam especial interesse as funções do tipo $\sum(a_j E_j + b_j T_j)$, em que a_j e b_j são pesos, ou custos, de antecipação e atraso, respectivamente. Estas funções encontram-se em problemas designados de *"earliness-tardiness"* (E/T). Por exemplo, para o problema de E/T mais geral, sem data de entrega comum, poderá ser necessária a existência de *"idle time"* num calendário óptimo [Baker, Scudder 1990].

Na tabela Tabela 6.4 apresentam-se expressões para algumas das mais frequentes funções objectivo relacionadas com datas de entrega, que constituem a notação para o campo γ do esquema de [Graham et al. 1979].

Objectivo	**Expressão**
Atraso máximo	$L_{\max} = \max_{j=1,2,\ldots,n} L_j$
Número de tarefas atrasadas	$\sum_{j=1}^{n} U_j$
Atraso total	$\sum_{j=1}^{n} T_j$
Soma ponderada dos atrasos	$\sum_{j=1}^{n} w_j T_j$
Earliness-tardiness	$\sum(a_j E_j + b_j T_j)$

Tabela 6.4 Algumas funções objectivo relacionadas com datas de entrega

Exemplo 6.2

Considere-se a instância e a sequência introduzidas no Exemplo 6.1. Na Tabela 6.5 apresentam-se os correspondentes valores dos critérios atraso máximo, número de tarefas atrasadas, atraso total, soma ponderada dos atrasos e *earliness-tardiness* ($\alpha_j = \beta_j = 0.5$).

Tarefa j	p_j	d_j	w_j	C_j	L_j	T_j	E_j	U_j	w_jT_j	$\alpha_j E_j + \beta_j T_j$
1	2	5	1	3	-2	0	2	0	0	1
2	4	7	6	7	0	0	0	0	0	0
3	1	11	2	1	-10	0	10	0	0	5
4	3	9	3	13	6	6	0	1	18	3
5	3	8	2	10	2	2	0	1	4	1
					$L_{max}=6$	$\sum T_j=8$		$\sum U_j=2$	$\sum w_jT_j=22$	$\sum(\alpha_j E_j+\beta_j T_j)=10$

Tabela 6.5 Valores de critérios relacionados com datas de entrega

Para os três objectivos a considerar no caso multiobjectivo tratado neste trabalho (minimização do atraso máximo, do número de tarefas atrasadas e da soma ponderada dos atrasos) serão, de seguida, apresentados os correspondentes problemas com um só objectivo.

6.2.3 Minimização do atraso máximo (1 | b | L_{max})

Para datas de entrega e datas de lançamento arbitrárias, este problema é NP-difícil [Sousa 1989], mesmo com relações de precedência entre as tarefas. Na Tabela 6.6 apresentam-se alguns dos casos particulares resolúveis em tempo polinomial.

O problema que será abordado no caso multiobjectivo apresenta as mesmas características do primeiro dos casos particulares apresentados, com datas de lançamento iguais a zero. O problema de minimização do atraso máximo é, nessas condições, resolúvel em tempo polinomial (pela regra EDD), pelo que o correspondente método de resolução poderá ser usado como algoritmo construtivo para obter soluções iniciais para o problema multiobjectivo.

Problema	Descrição	Método de resolução
$1 \mid r_j = r \mid L_{max}$	Datas de lançamento iguais ou zero	Regra de Jackson ou EDD.
$1 \mid r_j, d_j = d \mid L_{max}$	Datas de entrega iguais	Sequenciar as tarefas por ordem não decrescente das datas de lançamento.
$1 \mid r_j, p_j = 1 \mid L_{max}$	Tempos de processamento iguais	Extensão à regra de Jackson: conferir maior prioridade a tarefas disponíveis com menores datas de entrega.
$1 \mid r_j, pntm \mid L_{max}$	Preempção permitida	Extensão à regra de Jackson: interrupção do processamento sempre que uma tarefa mais urgente se torne disponível.

Tabela 6.6 Casos particulares do problema de minimização do atraso máximo, resolúveis em tempo polinomial

6.2.4 Minimização do número de tarefas atrasadas ($1 \mid b \mid \sum U_j$)

Para datas de entrega e datas de lançamento arbitrárias, este problema é NP-difícil [Lenstra et al. 1977]. Alguns dos casos particulares resolúveis em tempo polinomial são apresentados na Tabela 6.7. O caso particular com datas de lançamento iguais a zero é, conforme referido na secção anterior, o caso que será alvo de uma abordagem multiobjectivo. Este problema, com o objectivo único de minimização do número de tarefas atrasadas, é resolúvel em tempo polinomial, pelo que também o correspondente método de resolução, a regra de Moore, poderá ser usado como algoritmo construtivo para obter soluções iniciais para o problema multiobjectivo.

Problema	Descrição	Método de resolução
$1 \mid r_j = r \mid \sum U_j$	Datas de lançamento iguais ou zero	Regra de Moore [Moore 1968]: as tarefas são sequenciadas por ordem não decrescente das datas de entrega; percorrendo a lista, sempre que uma tarefa está atrasada, determina-se a tarefa com maior tempo de processamento, desde o início da lista até à tarefa actual, inclusive, e coloca-se essa tarefa no final da lista.
$1 \mid r_i < r_j \Rightarrow d_i \leq d_j \mid \sum U_j$	Datas de lançamento e entregas compatíveis	Generalização da regra de Moore [Lawler 1982].

Tabela 6.7 Casos particulares do problema de minimização do número de tarefas atrasadas, resolúveis em tempo polinomial

6.2.5 Minimização da soma ponderada dos atrasos ($1 \mid b \mid \sum w_j T_j$)

Este problema, mesmo sem datas de lançamento, é NP-difícil [Lawler 1977]. Um problema particular resolúvel em tempo polinomial é $1 \mid r_j,\ p_j=1 \mid \sum w_j T_j$, em que os tempos de processamento são todos iguais a 1. Neste caso, o problema reduz-se a um problema de afectação linear, no qual o custo de processar a tarefa j na posição k é: $+\infty$ se $k < r_j$, 0 se $r_j \leq k \leq d_j$, ou $w_j (k - d_j)$ se $k > d_j$.

Várias abordagens têm sido propostas para a resolução deste tipo de problemas, das quais referiremos algumas das mais relevantes.

Técnicas exactas

Para o caso sem pesos, em [Emmons 1969] são determinadas várias regras de dominância que restringem a pesquisa de uma solução óptima, permitindo fixar a posição de algumas tarefas ou estabelecer relações de precedência entre elas. Os resultados foram posteriormente estendidos ao caso com pesos em [Rinnooy Kan et al. 1975], e utilizados em abordagens de *branch-and- bound* ([Fisher 1976] e [Potts, van Wassenhove 1985]) e Programação Dinâmica ([Lawler 1977] e [Potts, van Wassenhove 1987]). Uma comparação computacional de diversos algoritmos exactos aplicados ao problema da soma ponderada dos atrasos, pode ser encontrada em [Abdul-Razaq et al. 1990].

Em [Rachamadugu 1987] é identificada uma nova propriedade que caracteriza tarefas adjacentes em sequências óptimas, e que está na base da heurística ATC, apresentada anteriormente. Na referência mais recente [Akturk, Yildirim 1998], é proposta uma nova regra de dominância, juntamente com uma revisão dos principais resultados teóricos na área das técnicas exactas.

Técnicas não-exactas

Em [Vepsalainen, Morton 1987] é demonstrado que a regra ATC é superior às restantes heurísticas de sequenciamento, e produz resultados perto do óptimo, para o problema da soma ponderada dos atrasos. Mais recentemente, em [Hosenback et al. 1999] é proposta uma heurística baseada na modificação das datas de entrega, para a qual são reportados resultados melhores do que os obtidos com a regra ATC.

A utilização de técnicas exactas e de regras de prioridade simples defronta-se com diversas dificuldades. Em particular, as regras simples não produzem boas soluções de forma consistente [Potts, van Wassenhove 1991] e os algoritmos exactos colocam fortes exigências computacionais em termos de tempo e memória, especialmente para instâncias com mais de 50 tarefas [Crauwels et al. 1998]. A aplicação de meta-heurísticas permite exactamente ir ao encontro destas dificuldades. As seguintes referências constituem exemplos da aplicação de meta-heurísticas ao problema da soma ponderada dos atrasos:

- [Potts, Van Wassenhove 1991], em que são propostas abordagens de *descent* e SA;
- [Madureira, Sousa 1996], com utilização de Pesquisa Local Aleatorizada (*"Randomized Local Search"*) e TS;
- [Crawels et al. 1998], com a aplicação de *descent*, SA, TA, TS e GA, com e sem multiarranque.

Em [Congram et al. 1999] é explorada uma nova abordagem, designada *dynasearch*, que combina *descent* e Programação Dinâmica. Na área das meta-heurísticas, os melhores resultados até ao momento foram obtidos em [Crawels et al. 1998] com TS com multiarranque, mas a abordagem *dynasearch* apresentou os melhores resultados globais reportados até ao momento.

6.3 Pesquisa local aplicada ao escalonamento numa máquina

Para além das referências apresentadas no caso do problema da soma ponderada dos atrasos, para outros problemas de escalonamento de tarefas numa máquina têm, igualmente, sido propostas abordagens baseadas em meta-heurísticas, que se enumeram na Tabela 6.8.

Problema	**Abordagem**	**Referências**
Custos de *setup* e soma ponderada dos atrasos, com custos de *setup* dependentes da sequência	TS	[Laguna et al. 1991 e Laguna, Glover 1993]
	GRASP	[Feo et al. 1996]
Custos de *setup* e atraso máximo	Redes neuronais	[Mendes, Aguilera 1998]
Flow time variance	GA	[Gupta et al. 1993]
E / T	GA, *descent* multi-arranque, GRASP e algoritmo híbrido de GA e *descent*	[Yagiura, Ibaraki 1995]
	TS com dois esquemas de representação de soluções	[James, Buchanan 1997, 1998]
	Algoritmo híbrido sequencial de SA e TS	[Almeida, Centeno 1998]
E / T com data de entrega comum	TS	[James 1997]
Atraso total	ACO	[Bauer et al. 1999]
Atraso total, com tempos de *setup* dependentes da sequência	SA	[Tan, Narasimhan 1997]
	GA e algoritmos miméticos	[França et al. 1999]
Tempo médio de conclusão	GA	[Liu, Tang 1999]
Time-lags positivos e negativos	TS	[Hurink, Keuchel 2001]

Tabela 6.8 Algumas referências de aplicações de meta-heurísticas em problemas de escalonamento de tarefas numa máquina

Para a abordagem multiobjectivo adoptada no âmbito desta dissertação, baseada em técnicas de pesquisa local, apresentam particular interesse as aplicações com um só objectivo nesta área, em especial de SA e TS. Esboçam-se, portanto, de seguida, os esquemas mais comuns de representação de soluções, algoritmos construtivos, estruturas de vizinhança e configuração de SA e TS.

6.3.1 Representação de soluções

Uma representação natural de uma solução para o problema de escalonamento de tarefas numa máquina, é dada pela *permutação* dos inteiros 1, ..., *n*, que define a ordem de processamento das tarefas. Esta representação é válida para os problemas concretos em que não há *"idle times"* e as tarefas são todas processadas imediatamente em sequência. Será, talvez, a representação mais vulgarizada, com utilização em diversas abordagens baseadas em meta-heurísticas. Mesmo para problemas que envolvam *"idle times"*, existem abordagens que utilizam esta representação numa primeira fase, e, posteriormente, a trabalham com algoritmos de inserção optimizada dos *"idle times"*.

Uma segunda representação, relativamente frequente, *binária*, é dada por um vector de *n* bits, em que cada bit representa o estado de atraso de uma tarefa: o bit *j* é 0 se a tarefa *j* está atrasada, e é 1 se não está atrasada. Em [James, Buchanan 1997] esta representação é utilizada no problema E/T, conjuntamente com uma heurística, não optimizadora, que transforma o vector binário numa sequência de tarefas. De forma independente, em [Crawels et al. 1998] esta representação é também utilizada, juntamente com uma heurística de descodificação para uma sequência de tarefas.

6.3.2 Algoritmos construtivos

As técnicas construtivas mais frequentes foram já introduzidas, sendo utilizadas regras de prioridade básicas ou compostas, bem como outras regras mais elaboradas (como a regra de Moore) e também a geração aleatória de soluções.

No caso das representações binárias, em [James, Buchanan 1997] as soluções iniciais são construídas aleatoriamente, enquanto em [Crawels et al. 1998] é apresentado um algoritmo construtivo específico para este tipo de representação.

6.3.3 Estruturas de vizinhança

Para a representação baseada em permutações, as estruturas de vizinhança mais comuns em problemas de escalonamento, em particular de escalonamento numa máquina, são [Barnes, Laguna 1991]:

- *Adjacent Pairwise Interchange* (API), em que duas tarefas imediatamente adjacentes são trocadas de posição. Pode ser definida pelo operador api_k, para $1 \le k \le n\text{-}1$, em que api_k troca de posição os elementos nas posições k e $k+1$:

$$api_k(p_1,\ldots,p_{k-1},p_k,p_{k+1},p_{k+2},\ldots,p_n) = (p_1,\ldots,p_{k-1},p_{k+1},p_k,p_{k+2},\ldots,p_n)$$

  Esta vizinhança tem dimensão n-1.

- *Swap* (*All Pairwise Interchange*), em que quaisquer duas tarefas podem ser trocadas de posição. Pode ser definida pelo operador $swap_{k,l}$, para $1 \le k < l \le n$, em que $swap_{k,l}$ troca de posição os elementos nas posições k e l:

$$swap_{k,l}(p_1,\ldots,p_{k-1},p_k,p_{k+1},\ldots,p_{l-1},p_l,p_{l+1},\ldots,p_n)$$

$$= (p_1,\ldots,p_{k-1},p_l,p_{k+1},\ldots,p_{l-1},p_k,p_{l+1},\ldots,p_n)$$

  Esta vizinhança tem dimensão $\binom{n}{2} = \frac{n(n-1)}{2}$.

- *Insert* (ou *Shift*), em que uma tarefa é removida da sua posição actual e colocada numa nova posição. Pode ser definida pelo operador $insert_{k,l}$, para $1 \le k \le n$, $1 \le l \le n$, $k \ne l$ e $l \ne k\text{-}1$, em que $insert_{k,l}$ desloca o elemento na posição k para a posição l:

$$insert_{k,l}(p_1,\ldots,p_{k-1},p_k,p_{k+1},\ldots,p_{l-1},p_l,p_{l+1},\ldots,p_n)$$

$$= (p_1,\ldots,p_{k-1},p_{k+1},\ldots,p_{l-1},p_l,p_k,p_{l+1},\ldots,p_n)$$

  A restrição $l \ne k\text{-}1$ evita a duplicação originada pelo facto de as aplicações de $insert_{a,a+1}$ e $insert_{a+1,a}$ conduzirem à mesma permutação.

  Esta vizinhança tem dimensão $(n-1)^2$.

São referidos na literatura diversos estudos comparativos do desempenho destas vizinhanças em problemas de escalonamento:

- em [Laguna, Glover 1993] a vizinhança *Insert* apresenta, de forma consistente, resultados superiores aos da vizinhança *Swap*;
- em [Tsubakitanni, Evans 1992] a vizinhança *Swap* apresenta, de forma consistente, resultados superiores aos da vizinhança API;
- em [Barnes, Laguna 1991, 1993] os melhores resultados são obtidos com uma combinação de *Insert* e *Swap*.

Algumas variantes destas vizinhanças têm sido usadas, considerando para os movimentos de *Swap* e *Insert* uma distância máxima ou mínima admissível entre as posições de tarefas em causa ([Madureira, Sousa 1996] e [Almeida, Centeno 1998]). Em [Crawels et al.

1998] a variação destas distâncias é usada com objectivos específicos de intensificação e diversificação.

Para a representação binária, quer em [James, Buchanan 1997], quer em [Crawels et al. 1998], é sugerida uma vizinhança baseada na mudança de estado de atraso de apenas uma tarefa.

As sub-vizinhanças têm sido frequentemente aplicadas ([Reeves 1993], [Madureira, Sousa 1996], [James, 1997], [James, Buchannan 1997, 1998] e [Crawels et al. 1998]), com o objectivo de permitir explorar mais regiões do espaço de soluções, com o esforço computacional disponível, através de uma menor concentração da pesquisa. Procura-se, assim, que, com maior probabilidade, a pesquisa não fique bloqueada em mínimos locais.

6.3.4 *Simulated Annealing*

Para estes problemas, as implementações de SA diferenciam-se sobretudo pela diversidade de esquemas de arrefecimento. Nas referências apontadas anteriormente que utilizam SA, são adoptados em geral esquemas de arrefecimento genéricos, não adaptados especificamente ao problema de escalonamento numa máquina.

Em [Tan, Naramsihan 1997] é utilizado um esquema idêntico ao referido no capítulo relativo às meta-heurísticas, na introdução ao SA, sendo os parâmetros configurados da seguinte forma:

- O comprimento do *"plateau"* é testado com valores de 1, 50 e 100, por forma a investigar a sua influência na solução final.
- A temperatura inicial é calculada por forma a que a correspondente probabilidade de aceitar o valor mais elevado de deterioração, encontrado numa série de execuções preliminares, seja 0.5.
- A temperatura final obtém-se, de forma idêntica, a partir de uma probabilidade de aceitar o valor mais reduzido de deterioração, definida como 0.00005.
- A taxa de arrefecimento foi seleccionada a partir de uma análise do esforço computacional necessário para baixar a temperatura do valor inicial para o valor final, para diversas taxas possíveis; a selecção recaía sobre o valor a partir do qual se registava um forte aumento do tempo de computação, no caso, 0.995.

Em [Almeida, Centeno 1998] são seguidas as indicações gerais referidas em [Johnson et al. 1989]. Em particular, a temperatura inicial é escolhida por forma a que a probabilidade de aceitação inicial esteja na casa dos 4%, o factor de arrefecimento seleccionado é 0.95 e a

paragem do algoritmo é efectuada quando, em k níveis de temperatura sucessivos, após a última melhoria de solução, não são aceites mais do que 2% de soluções.

Em [Crawels et al. 1998] é utilizada a implementação de [Potts, Van Wassenhove 1991]:

- A temperatura t_k em cada nível k ($1 \leq k \leq l$) é obtida a partir de uma probabilidade de aceitação K_k, de aceitar um aumento de 1% no melhor valor da função objectivo encontrado até ao momento Z, ou seja,

$$t_k = -\frac{0.01Z}{\ln K_k}.$$

- A probabilidade de aceitação decresce geometricamente:

$$K_k = aK_{k-1}, a = \left(\frac{K_l}{K_1}\right)^{\frac{1}{l-1}},$$

em que K_1 e K_l são, respectivamente, as probabilidades de aceitação inicial e final, e l é o número de níveis de temperatura.

Para K_1 e K_l, os valores usados em [Crawels et al. 1998] são $K_1 = 0.99$, $K_l = 0.001$. l é proporcional ao número de tarefas, tendo sido analisados os seguintes valores: $3n$ e $6n$ no caso simples; $3n/5$ e $6n/5$ no caso multiarranque com 5 iterações. O desempenho do algoritmo manifesta insensibilidade aos valores de K_1 e K_l, desde que estes sejam elevados para K_1 ($K_1 \geq 0.5$), e reduzidos para K_l ($K_l \leq 0.001$).

6.3.5 Pesquisa Tabu

No caso das implementações de TS, os principais aspectos que requerem configuração são o comprimento da lista tabu, os atributos a guardar na lista, o critério tabu e o critério de aspiração.

Em [Madureira, Sousa 1996] são considerados vários tamanhos para a lista tabu: 0, 4, 7 e 10 (com a opção 0 pretende-se verificar a influência da utilização da lista). Não foram detectadas diferenças significativas de desempenho, sendo no entanto recomendado um valor próximo de 7. A vizinhança utilizada é exclusivamente do tipo *swap*, sendo considerados alternativamente dois tipos de atributos para a lista: tarefas individuais ou pares de tarefas. No primeiro caso, o critério tabu consiste em inibir qualquer movimento que se realize sobre uma das tarefas presentes na lista. No segundo caso, são considerados tabu todos os movimentos que envolvam nova troca de qualquer par de tarefas presente na lista. O critério de aspiração

utilizado consiste na aceitação de um movimento tabu no caso de este conduzir a um novo melhor valor global para a função objectivo.

A utilização de duas representações distintas num conjunto de trabalhos relacionados, apresentados em [James, Buchanan 1997, 1998] e [James 1997], implica a consideração de esquemas de configuração distintos:

- Para representações baseadas em permutações, com problemas de pequena dimensão (8 a 15 tarefas) foram considerados tamanhos de lista entre 1 e 8, não se tendo registado influência da sua variação no desempenho do algoritmo. Para problemas de maior dimensão (250 tarefas) é recomendado um tamanho de lista de 15.

  A vizinhança utilizada combina movimentos de *swap* e *insert*. Os atributos utilizados consistem na tarefa movimentada, no caso de *insert*, e apenas numa das tarefas trocadas, no caso de *swap*. É também guardada informação da nova posição das tarefas. O critério tabu consiste em inibir os movimentos que envolvam modificações das posições de tarefas presentes na lista.

- Para representações binárias, com problemas de pequena dimensão, foram novamente considerados tamanhos de lista entre 1 e 8, tendo-se verificado influência da sua variação no desempenho do algoritmo. Para problemas de maior dimensão (250 tarefas) é recomendado um tamanho de lista de 1.

  A vizinhança utilizada, conforme já referido a respeito de estruturas de vizinhança utilizadas em problema de escalonamento de uma máquina, consiste na inversão do estado de atraso de uma única tarefa. Na lista tabu, são armazenadas as tarefas cujo estado de atraso foi invertido mais recentemente. O critério tabu consiste em inibir movimentos envolvendo as tarefas presentes na lista.

Em ambas as representações, o critério de aspiração baseia-se, novamente, na aceitação de um movimento tabu que conduza a um melhor valor global para a função objectivo.

Em [Almeida, Centeno 1998] são utilizadas listas de comprimentos 3, 5 e 7 para problemas de, respectivamente, 10, 20 e 50 tarefas. As vizinhanças utilizadas são do tipo *swap*, sendo os atributos da lista constituídos pelos pares de tarefas trocadas mais recentemente. O critério tabu consiste em impedir a inversão das trocas de pares de tarefas presentes na lista, sendo o critério de aspiração idêntico ao dos casos anteriores.

Em [Hurink, Keuchel 2001] é recomendada uma lista de comprimento função do número de tarefas, dada por $\sqrt{2n}-2$. Tal como na referência anterior, a vizinhança é baseada em movimentos do tipo *swap* e os atributos da lista tabu são os pares de tarefas trocadas mais recentemente. Para estes atributos, o critério tabu inibe movimentos que envolvam a troca de duas tarefas que formem uma das entradas da lista. Não é referida a utilização de um critério de aspiração.

Em [Crawels et al. 1998] são novamente usadas duas representações. São recomendados comprimentos de lista de 7, para a representação baseada em permutações, e de $n/3$, para a representação binária. Não foram encontradas diferenças de desempenho significativas, que motivassem a utilização de tamanhos de lista superiores a 7. A vizinhança utilizada com a representação baseada em permutações é do tipo *swap*. Os atributos guardados na lista tabu consistem apenas numa das tarefas trocadas no movimento e do respectivo valor da função objectivo. São considerados tabu os movimentos que envolvam tarefas presentes na lista. O critério de aspiração utilizado permite a aceitação de um movimento tabu, se o respectivo valor da função objectivo for melhor do que os valores correspondentes aos itens da lista que o tornam tabu.

No caso da representação binária, a vizinhança é idêntica à referida nos trabalhos de [James, Buchanan 1997, 1998] e [James 1997]. Os atributos são constituídos pelas tarefas cujo estado de atraso foi invertido mais recentemente e o seu novo estado. O critério tabu inibe a realização de movimentos que devolvam ao estado original as tarefas presentes na lista. O critério de aspiração é aqui, novamente, a aceitação de um movimento tabu que conduza a um melhor valor global para a função objectivo.

Convirá, ainda, no âmbito da TS, destacar a utilização de estratégias de intensificação e diversificação. Em particular no trabalho apresentado em [Crawels et al. 1998], são usados alguns mecanismos com estes objectivos:

- Para limitar o inconveniente de não serem detectados alguns movimentos bons, que resulta da utilização de sub-vizinhanças, é usado um mecanismo de intensificação quando, à execução de um movimento de melhoria, se segue, de imediato, a execução de um movimento de deterioração. Essa intensificação é realizada através da aplicação de um *descent* com uma vizinhança do tipo API, partindo da solução anterior à degradação.

- A diversificação é particularmente útil em problemas de escalonamento numa máquina, semelhantes ao da soma ponderada dos atrasos, em que um número elevado de tarefas não estão atrasadas, sendo inconsequentes muitos movimentos que envolvem essas tarefas. Assim, após $n/3$ iterações apenas com movimentos

inconsequentes, é usado um mecanismo de diversificação, que consiste em seleccionar a melhor de entre 3 trocas aleatórias, entre uma das primeiras $n/4$ tarefas e uma das últimas $n/4$ tarefas.

Com a representação binária, após 10 movimentos inconsequentes, consideram-se apenas movimentos que alteram o estado das tarefas de não-atraso para atraso, e apresentam, consequentemente, uma maior probabilidade de deterioração da solução, uma vez que passará a estar atrasada mais uma tarefa.

6.4 Escalonamento multiobjectivo numa máquina

6.4.1 Introdução

Em [Erghott, Gandibleux 2000] é apresentada uma extensa bibliografia anotada sobre optimização combinatória multiobjectivo, que inclui uma secção dedicada ao escalonamento multiobjectivo. Os autores referidos apontam a existência nos anos mais recentes de um interesse constante por problemas de escalonamento multiobjectivo, justificada pela natureza multicritério dos problemas práticos de escalonamento. Um *survey* recente [T'Kindt, Billaut 1999] classifica cerca de cem problemas desta área.

Em [Chen, Bulfin 1993] são apontadas as abordagens multiobjectivo mais frequentes na área do escalonamento, caracterizadas para o caso biobjectivo, mas com uma extensão imediata a casos com um número superior de objectivos:

- *Considerar objectivos secundários*. Num caso biobjectivo, um objectivo é considerado primário e o outro secundário. Procura-se encontrar a melhor solução para o objectivo secundário de entre todas as soluções óptimas para o objectivo primário. Uma possível extensão à notação de [Graham et al. 1979] passaria por denotar estes problemas por $\alpha \mid \beta \mid \gamma_2/\gamma_1$, em que γ_1 seria o critério primário e γ_2 o critério secundário.
- *Gerar todas as soluções eficientes* e permitir ao agente de decisão analisar os compromissos entre as várias soluções. Um problema biobjectivo, com os objectivos γ_1 e γ_2, seria denotado por $\alpha \mid \beta \mid \gamma_1, \gamma_2$.
- *Utilizar funções de ponderação* que reflectem o compromisso, entre os vários objectivos, expresso pelo agente de decisão. Um problema biobjectivo, com os objectivos γ_1 e γ_2, e uma função de ponderação $f(\gamma_1, \gamma_2)$, seria denotado por $\alpha \mid \beta \mid f(\gamma_1, \gamma_2)$.

6.4.2 O problema considerado

Também no âmbito das abordagens multiobjectivo, os problemas de escalonamento numa máquina têm merecido especial atenção, existindo um conjunto razoavelmente extenso de referências que podem ser encontradas na bibliografia anotada de [Erghott, Gandibleux 2000].

Um estudo da complexidade dos problemas de escalonamento multiobjectivo numa máquina é apresentado em [Chen, Bulfin 1993], para os três tipos de abordagens referidos na subsecção anterior e para todas as combinações dos seguintes objectivos: *atraso máximo*, *soma dos tempos de conclusão*, *número de tarefas atrasadas*, *soma dos atrasos* e as versões ponderadas dos últimos três.

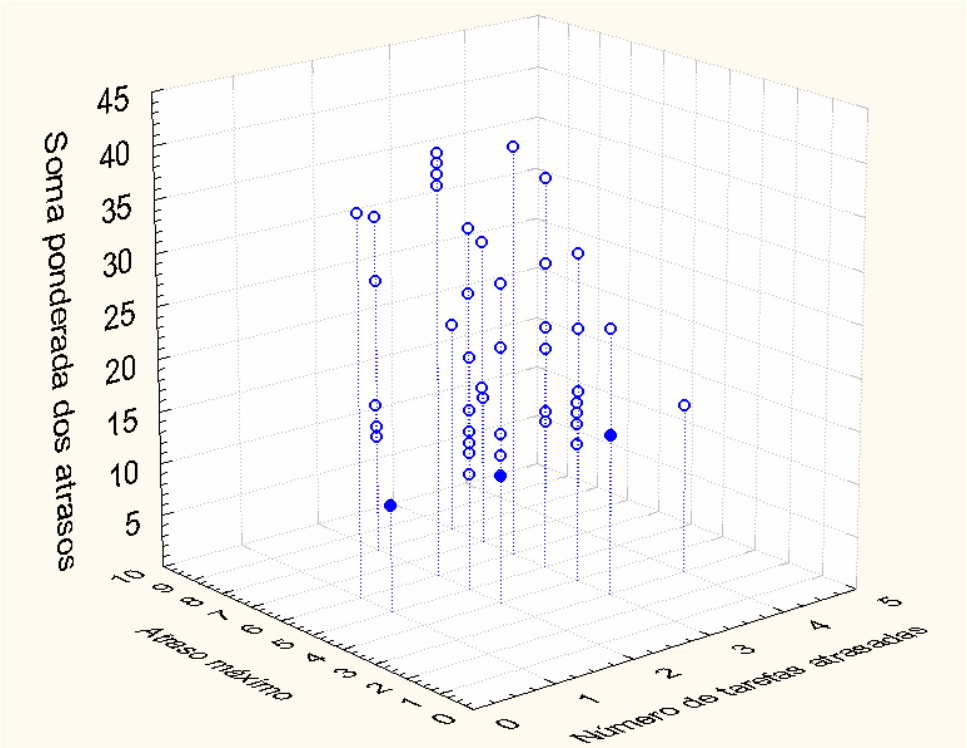
Muitos dos problemas de escalonamento multiobjectivo numa máquina, que têm sido alvo de estudo, são biobjectivo. O problema tratado no presente trabalho considera três objectivos: o *número de tarefas atrasadas*, o *atraso máximo* e a *soma ponderada dos atrasos*. Com base na notação proposta em [Chen, Bulfin 1993], este problema seria denotado $1|\ |\Sigma U_j,\ L_{max},\ \Sigma w_j T_j$.

Os três objectivos considerados constituem, na prática, três das mais frequentes formas de avaliação do atraso de um escalonamento, e apresentam perspectivas distintas dessa mesma avaliação. Na revisão de literatura realizada no âmbito desta dissertação, não foram encontradas quaisquer referências a este problema.

O problema é NP-difícil: sendo um dos problemas individuais NP-difícil (o que sucede com a minimização da soma ponderada dos atrasos), o problema multiobjectivo será também NP-difícil [Chen, Bulfin 1993]. A utilização de meta-heurísticas multiobjectivo para a abordagem deste problema justifica-se, portanto, plenamente.

Exemplo 6.3

Considere-se a instância introduzida no Exemplo 6.1. Na Figura 6.1 apresenta-se o espaço dos objectivos para a instância considerada, que compreende 120 (5!) soluções. A cheio assinalam-se as soluções não-dominadas no espaço dos objectivos (ΣU_j, L_{max}, $\Sigma w_j T_j$): (1, 5, 10), (2, 4, 12) e (3, 3, 15).



6.4.3 Aplicação de meta-heurísticas multiobjectivo

Na pesquisa de literatura realizada, foram encontradas apenas três referências de aplicação de meta-heurísticas a problemas de escalonamento multiobjectivo numa máquina:

- [Tamaki et al. 1994], com GA.

  Trata-se de um problema no contexto de um problema de escalonamento mais complexo, envolvendo o *"hot rolling process"* de uma unidade de produção de aço. Os objectivos considerados são específicos à situação em análise: índice de complexidade da sequência, quantidade total de combustível, tempo total de aquecimento.

O algoritmo utilizado baseia-se na abordagem de [Schaffer 1985], apresentada em capítulo anterior, e na aplicação de uma estratégia designada *"Pareto Reservation Strategy"*, que consiste em preservar na população todos os indivíduos não-dominados de uma geração. É adicionalmente utilizada pesquisa local simples com uma vizinhança do tipo *swap* para melhoria das soluções obtidas com GA.

- [Karasakal, Köksalan 2000], com SA.

  Dois problemas biobjectivo NP-difíceis são estudados: a minimização da soma dos tempos de conclusão e do atraso negativo (*earliness*) máximo; e a minimização da soma dos tempos de conclusão e do número de tarefas atrasadas.

  Para o primeiro problema, a abordagem utilizada consiste na realização de uma execução de SA, para cada valor de atraso negativo máximo.

  No segundo problema, é inicialmente determinado o ponto ideal: o mínimo número de tarefas atrasadas é determinado pela regra de Moore, enquanto o valor mínimo da soma dos tempos de conclusão é dado pela regra SPT. Para cada número de tarefas atrasadas, (incrementado unitariamente, de uma forma iterativa) é realizada uma execução de SA. Com o objectivo de gerar todas as soluções eficientes, é utilizada como função objectivo uma função escalarizante de Tchebycheff ponderada, que minimiza a distância ao ponto ideal. Os pesos dessa função são variados dinamicamente durante a execução do algoritmo. As gamas dos dois objectivos são equalizadas.

- [Morita et al. 2001], com GA.

  O problema abordado é a minimização da soma dos tempos de conclusão e do atraso máximo.

  O algoritmo utiliza quatro estratégias principais: manter na população todas as soluções com a classificação mais elevada, de acordo com o método de [Goldberg 1989]; manter na população algumas soluções de boa qualidade relativamente a cada objectivo, com base na abordagem de [Schaffer 1985]; seleccionar, por comparação de pares, com o objectivo de manter a diversidade de indivíduos; incluir na população soluções de boa qualidade relativamente a cada objectivo (*"Seeding Strategy"*), determinadas por algoritmos exactos ou heurísticas apropriadas para os problemas com um só objectivo.

Com o *framework* desenvolvido neste trabalho serão aplicadas a este problema, as meta-heurísticas multiobjectivo baseadas em pesquisa local PSA e MOTS*, e um conjunto de

variantes destas meta-heurísticas, com utilização de vizinhanças variáveis e listas de candidatos, e uma abordagem híbrida e paralela de alto nível.

As meta-heurísticas base, bem como as diversas variantes, foram já descritas em capítulo anterior. A aplicação dessas meta-heurísticas a este problema em concreto será o objecto da secção seguinte, no âmbito da descrição da aplicação do *framework*. Por fim, os aspectos de parametrização dos algoritmos, serão apresentados no capítulo relativo aos testes computacionais.

6.5 Aplicação do *framework* desenvolvido ao escalonamento multiobjectivo numa máquina

6.5.1 Introdução

Apresenta-se nesta secção a forma como foi utilizado o *framework* desenvolvido, na aplicação de abordagens de pesquisa local multiobjectivo ao problema em estudo. Esta aplicação exige a definição de dois tipos de elementos:

- os dados que definem uma instância do problema a resolver;
- as classes particulares a derivar das classes do *framework*:
  - a solução do problema e as respectivas avaliações;
  - as estruturas de vizinhança (movimentos e atributos, geradores e avaliação);
  - os algoritmos construtivos (incrementos e geradores de incrementos).

De modo a permitir uma utilização configurável do *framework*, e com o objectivo estrito de realizar testes computacionais, foi construído um interface dedicado que permite especificar uma configuração pretendida, quer ao nível das classes particulares atrás referidas, quer ao nível das técnicas algorítmicas a aplicar. O facto de este interface ter um carácter meramente temporário e ter sido desenvolvido exclusivamente para a realização dos testes computacionais, leva a que a sua descrição não seja incluída neste texto.

O conjunto de técnicas algorítmicas a aplicar, de entre as possibilidades genéricas consagradas no *framework* e descritas no capítulo anterior, serão apresentadas no Capítulo 7, relativo aos testes computacionais. Nesta secção apenas se descreverão o formato dos dados de uma instância de problema e as classes particulares a derivar das classes do *framework*.

6.5.2 Dados do problema

Os únicos objectos a considerar explicitamente numa instância do problema, são as tarefas. Estas são caracterizadas, conforme já referido, por um tempo de processamento, um peso e uma data de entrega. A esta descrição de uma instância do problema corresponderá, no *framework*, a definição de um índice, relativo às tarefas, e três parâmetros, indexados sobre as tarefas e relativos às três características apontadas.

O ficheiro com os dados do problema correspondente a esta descrição apresenta o formato da Figura 6.2. Os parâmetros serão carregados na sequência identificada na figura: o primeiro será o tempo de processamento; o segundo, a data de entrega; e o terceiro, o peso.

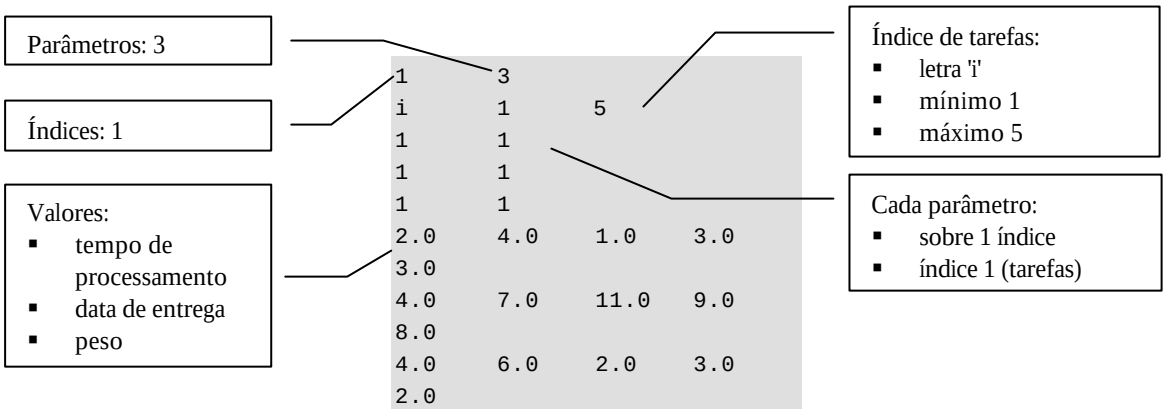


Figura 6.2 Formato para o ficheiro sequencial de dados

6.5.3 Soluções e avaliação

Soluções

Uma vez que, pelas suas características, o problema considerado é um problema de sequenciamento, as soluções serão implementadas com uma representação baseada em permutações. A Figura 6.3 apresenta a estrutura de classes para a *solução para problemas de sequenciamento* (*SeqSolution*).

Esta classe é uma especialização da classe *solução* (*Solution*). Os seus atributos e métodos específicos são apresentados na Tabela 6.9.

Um aspecto essencial na configuração das soluções é a sua associação a *avaliadores*. A criação dessa associação deverá ser realizada através da inserção directa dos avaliadores no respectivo vector de avaliadores. A referência deste vector é disponibilizada pelo método *GetEvaluators*.

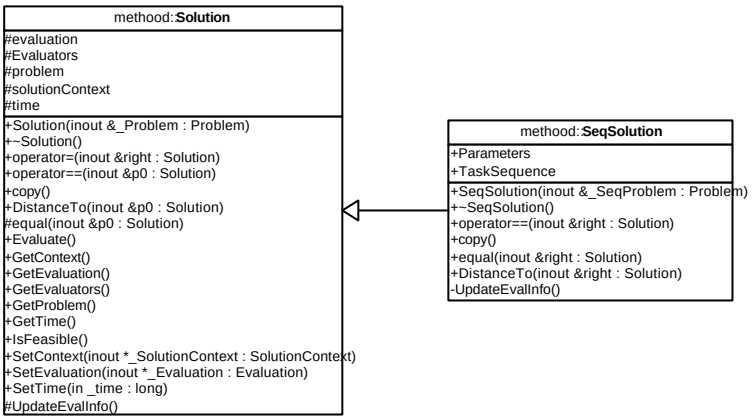


Figura 6.3 *Solução para problemas de sequenciamento* - diagrama de classes

Atributo	**Descrição**
TaskSequence	Vector contendo a sequência das tarefas
Parameters	Vector de apontadores para os vectores com os valores dos parâmetros
Método	**Descrição**
SeqSolution	Construtor da classe. Constrói *Parameters* a partir dos dados do problema, obtidos a partir de *Problem*.
~SeqSolution	Destrutor da classe.
Operator==	Testa a igualdade de soluções, através da igualdade dos vectores de sequência de tarefas.
Copy	Cria uma nova solução e utiliza *equal* para a tornar igual a si própria.
Equal	Torna uma outra solução igual à própria, ao nível dos vectores de sequência de tarefas e das avaliações das soluções.
DistanceTo	Função de distância entre uma outra solução e a própria. A distância implementada consiste do número de tarefas com estado de atraso diferente entre as duas soluções.
UpdateEvalInfo	Actualiza o interface com os avaliadores. Em particular, actualiza as referências para a sequência de tarefas e para os parâmetros.

Tabela 6.9 *Solução para problemas de sequenciamento* - atributos e métodos específicos

Avaliação de soluções

Define-se uma classe *avaliador de soluções* para cada um dos objectivos considerados: soma ponderada dos atrasos, número de tarefas atrasadas e atraso máximo. Na Figura 6.4 é apresentada a estrutura de classes para estes avaliadores.

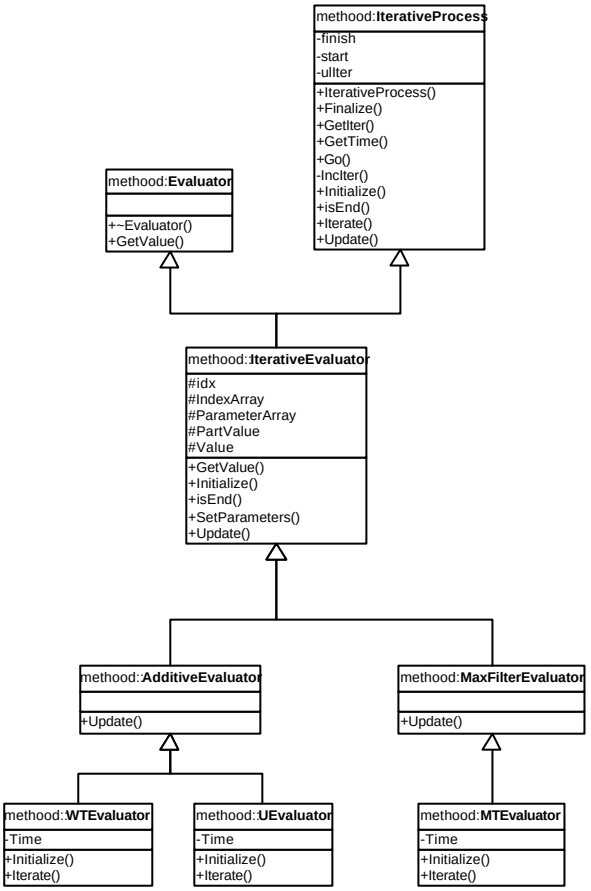


Figura 6.4 *Avaliação de soluções para problemas de sequenciamento* - diagrama de classes

Para a definição destas classes, que partilham muitos aspectos comuns de comportamento e informação, foi construída uma hierarquia de herança, que se descreve brevemente, de seguida:

- A classe *IterativeEvaluator* implementa uma função de avaliação que, de forma iterativa, trabalha parâmetros (*Parameters*) contidos em vectores indexados pelos elementos de um vector de índices (*IndexArray*). Em cada iteração, o avaliador calcula e disponibiliza um valor parcial (*PartValue*) que irá contribuir, de forma explicitada pelas classes derivadas, para um valor de avaliação final (*Value*).

  Devido ao seu comportamento iterativo, e com o objectivo de estruturar e simplificar a implementação de classes deste tipo, a classe é derivada de *IterativeProcess*.

- A classe *AdditiveEvaluator* constitui uma especialização de *IterativeEvaluator* em que *Value* é a soma de todos os *PartValue*. O método *Update* realiza esta operação.
- A classe *MaxFilterEvaluator* constitui uma especialização de *IterativeEvaluator* em que *Value* é o maior de todos os *PartValue*. O método *Update* é responsável pela realização deste processo de filtragem.

A classe correspondente ao objectivo soma ponderada dos atrasos é *WTEvaluator* (WT são as iniciais da designação inglesa *Weighted Tardiness*). Os seus atributos e métodos específicos são apresentados na Tabela 6.10.

Atributo	**Descrição**
Time	Tempo total de processamento acumulado até à iteração actual.
Método	**Descrição**
Initialize	Chama a função de inicialização da classe base e inicializa o tempo total de processamento.
Update	Verifica se a tarefa actual é realizada com atraso e, no caso positivo, calcula o valor parcial, multiplicando o atraso pelo peso unitário. Actualiza o tempo total de processamento.

Tabela 6.10 *WTEvaluator* - atributos e métodos específicos

A classe correspondente ao objectivo número de tarefas atrasadas é *UEvaluator* (utiliza-se a designação U, que na notação deste tipo de problemas identifica o estado de atraso de uma tarefa). Dos seus atributos e métodos específicos, apenas *Update* difere dos apresentados para *WTEvaluator*, motivo pelo qual é o único a ser apresentado na Tabela 6.11.

Método	**Descrição**
Update	Verifica se a tarefa actual é realizada com atraso. No caso positivo, o valor parcial será 1; no caso negativo, será 0. Actualiza o tempo total de processamento.

Tabela 6.11 *UEvaluator* - método *Update*

A classe correspondente ao objectivo atraso máximo é *MTEvaluator* (sendo MT as iniciais da designação inglesa *Maximum Tardiness*). Tal como *UEvaluator*, dos seus atributos e métodos específicos, apenas *Update* difere dos apresentados para *WTEvaluator*, sendo apresentado na Tabela 6.12.

Método	Descrição
Update	Verifica se a tarefa actual é realizada com atraso. No caso positivo, o valor parcial será esse atraso; no caso negativo, será 0. Actualiza o tempo total de processamento.

Tabela 6.12 *MTEvaluator* - método *Update*

6.5.4 Algoritmos construtivos

Incrementos

Para a representação de solução considerada, uma identificação natural de um *incremento para um problema de sequenciamento* (*SeqIncrement*) será o índice da tarefa a acrescentar ao vector de sequência de tarefas. A Figura 6.5 apresenta a correspondente estrutura de classes.

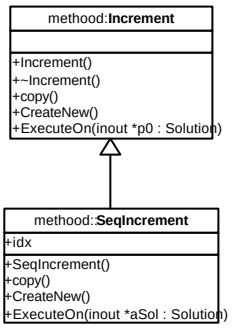


Figura 6.5 *Incremento para problemas de sequenciamento* - diagrama de classes

Esta classe constitui uma especialização da classe *incremento* (*Increment*). Os seus atributos e métodos específicos são apresentados na Tabela 6.13.

Atributo	**Descrição**
Idx	Índice da tarefa a acrescentar ao vector de sequência de tarefas
Método	**Descrição**
SeqIncrement	Construtor da classe.
Copy	Cria um novo incremento com o mesmo índice de tarefa.
CreateNew	Cria um novo incremento.
ExecuteOn	Acrescenta um novo elemento com o seu índice de tarefa no fim do vector de sequência de tarefas de uma solução.

Tabela 6.13 *Incremento para problemas de sequenciamento* - atributos e métodos específicos

Geradores de incrementos

São considerados quatro geradores de incrementos, correspondentes a quatro regras de prioridade: aleatória, regra EDD, regra de Moore e regra ATC. A Figura 6.6 apresenta a estrutura de classes para estes geradores.

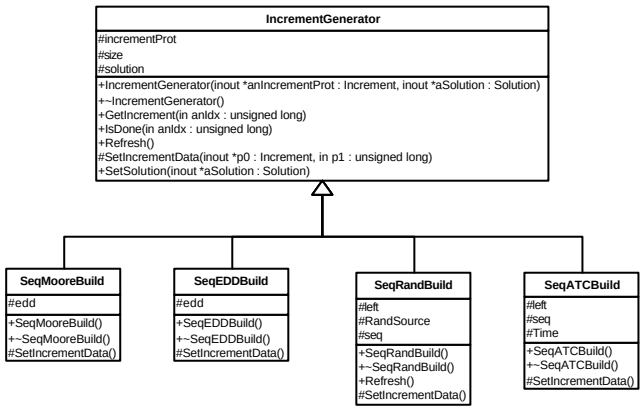


Figura 6.6 *Geradores de incrementos para problemas de sequenciamento* - diagrama de classes

Os atributos dos métodos das classes derivadas são omitidos para maior clareza da figura, e uma vez que todos constituem concretizações de métodos abstractos da classe base, na qual os atributos estão explicitados.

A classe correspondente à regra aleatória é *SeqRandBuild*. Os seus atributos e métodos específicos são apresentados na Tabela 6.14.

Atributo	**Descrição**
Left	Vector com os índices das tarefas ainda não acrescentadas à solução.
RandSource	Gerador de números aleatórios.
seq	Vector com a sequência de tarefas gerada.
Método	**Descrição**
SeqRandBuild	Construtor da classe. Inicializa a dimensão do gerador e o vector com a sequência de tarefas gerada. Chama *Refresh*.
~SeqRandBuild	Destrutor da classe.
Refresh	Gera nova sequência aleatória de tarefas. Vai escolhendo aleatoriamente tarefas de *left*, que acrescenta em *seq*.
SetIncrementData	O parâmetro índice deste método indica a posição do vector *seq* em que se vai obter o índice de tarefa que será incluído no incremento.

Tabela 6.14 *SeqRandBuild* - atributos e métodos específicos

A classe que implementa a regra EDD é *SeqEDDBuild*, cujos atributos e métodos específicos são apresentados na Tabela 6.15.

Atributo	**Descrição**
Edd	Vector cujos elementos contêm o índice e a data de entrega das tarefas.
Método	**Descrição**
SeqEDDBuild	Construtor da classe. Inicializa a dimensão do gerador e o vector *edd*. Ordena o vector *edd* por data de entrega não decrescente.
~SeqEDDBuild	Destrutor da classe. Invoca os destrutores dos componentes de *edd*.
SetIncrementData	O parâmetro índice deste método indica a posição do vector *edd* em que se vai obter o índice de tarefa que será incluído no incremento.

Tabela 6.15 *SeqEDDBuild* - atributos e métodos específicos

A regra de Moore é implementada com a classe *SeqMooreBuild*. Na Tabela 6.16 são apresentados os seus atributos e métodos específicos.

Atributo	**Descrição**
Edd	Vector cujos componentes contêm o índice, a *due date* e o tempo de processamento de cada tarefa.
Método	**Descrição**
SeqMooreBuild	Construtor da classe. Inicializa a dimensão do gerador e o vector *edd*. Ordena o vector *edd* por *due date* não decrescente. De seguida, reordena-o, aplicando a regra de Moore.
~SeqMooreBuild	Destrutor da classe. Invoca os destrutores dos componentes de *edd*.
SetIncrementData	O parâmetro índice deste método indica a posição do vector *edd* em que se vai obter o índice de tarefa que será incluído no incremento.

Tabela 6.16 *SeqMooreBuild* - atributos e métodos específicos

Por último, à regra ATC corresponde *SeqATCBuild*, cujos atributos e métodos específicos são apresentados na Tabela 6.17.

Atributo	**Descrição**
Left	Vector com os índices das tarefas ainda não acrescentadas à solução.
Seq	Vector com a sequência de tarefas gerada.
Time	Acumulador do tempo de processamento.
Método	**Descrição**
SeqATCBuild	Construtor da classe. Inicializa a dimensão do gerador e o vector com a sequência de tarefas gerada. Coloca todas as tarefas em *left*. Iterativamente, determina a que tem o máximo índice de prioridade de acordo com a regra ATC, retira-a de *left* e acrescenta-a em *seq*.
~SeqATCBuild	Destrutor da classe.
SetIncrementData	O parâmetro índice deste método indica a posição do vector *seq* em que se vai obter o índice de tarefa que será incluído no incremento.

Tabela 6.17 *SeqATCBuild* - atributos e métodos específicos

6.5.5 Vizinhanças

Movimentos e atributos

Neste trabalho, consideram-se dois tipos de movimentos: um movimento de troca de tarefas, correspondente a vizinhanças do tipo API e do tipo *swap*, e um movimento de deslocação de tarefa, correspondente à vizinhança do tipo *insert*. A representação genérica adoptada para os movimentos consiste em conjuntos de mudanças de posições de tarefas:

- Um movimento de *insert* é descrito através da identificação da tarefa deslocada, da sua posição inicial e da sua posição final. Em rigor, esta informação não descreve completamente o impacto do movimento sobre a solução, uma vez que as tarefas entre as duas posições são também deslocadas de uma posição. No entanto, é suficiente, quer para executar o movimento sobre uma dada solução, quer para a aplicação dos critérios tabu utilizados.
- Um movimento de troca de tarefas é descrito através da identificação das tarefas deslocadas, e das respectivas posições inicial e final.

A Figura 6.7 apresenta a estrutura de classes para esta representação de movimentos e atributos.

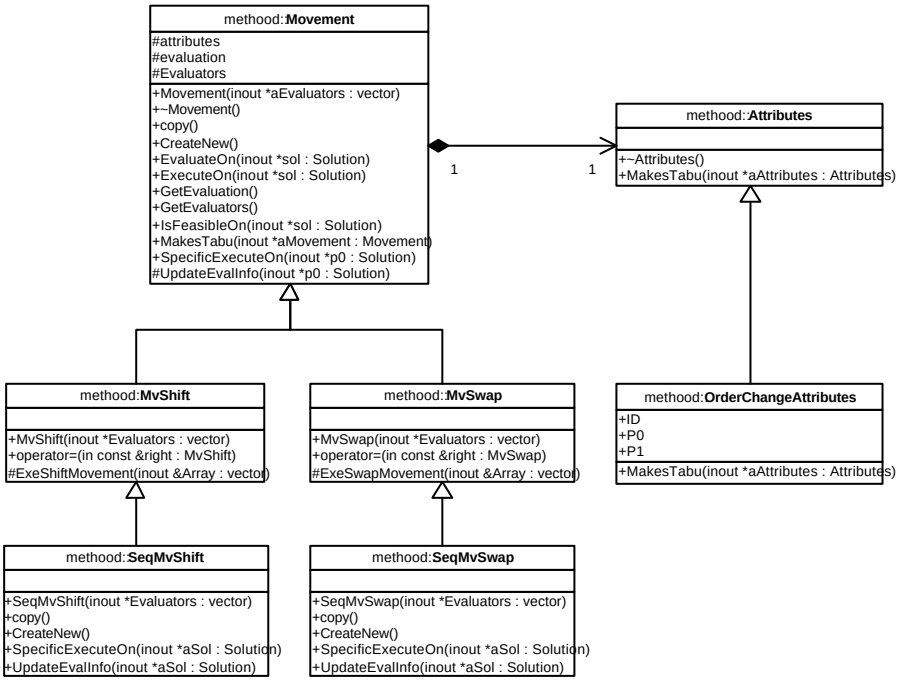


Figura 6.7 *Movimentos para problemas de sequenciamento* - diagrama de classes

Para a definição destas classes foi construída uma hierarquia de herança, que considera, para os movimentos referidos, um nível abstracto e geral (*MvShift* e *MvSwap*), e um

nível concreto e particular (*SeqMvShift* e *SeqMvSwap*) que implementa um interface específico às soluções para problemas sequenciais. O critério tabu implementado inibe a realização de movimentos com tarefas presentes nos atributos dos movimentos que compõem a lista tabu. A classe de atributos *OrderChangeAttributes* agrega toda a informação relativa aos movimentos considerados e estabelece o critério tabu referido. Os seus atributos e métodos específicos são apresentados na Tabela 6.18.

Atributo	**Descrição**
ID	Vector dos índices das tarefas com posições alteradas.
P0	Vector com as posições iniciais das tarefas com posições alteradas.
P1	Vector com as posições finais das tarefas com posições alteradas.
Método	**Descrição**
MakesTabu	Verifica, sobre os atributos recebidos como parâmetro, o critério tabu enunciado: inibe a realização de movimentos com tarefas presentes nos atributos dos movimentos que compõem a lista tabu.

Tabela 6.18 *Atributos de movimentos* - atributos e métodos específicos

As classes para definição dos movimentos do tipo *swap* possuem apenas métodos, que são descritos na Tabela 6.19, para o nível abstracto, e na Tabela 6.20, para o nível concreto.

Método	**Descrição**
MvSwap	Construtor da classe. Cria e inicializa os atributos.
operator=	Torna iguais dois movimentos (atributos e avaliações).
ExeSwapMovement	Executa um *swap* sobre dois elementos de um vector de inteiros, com base na informação de posições dos atributos.

Tabela 6.19 *MvSwap* - métodos específicos (nível abstracto)

Método	Descrição
SeqMvSwap	Construtor da classe.
CreateNew	Cria um novo movimento.
Copy	Cria um novo movimento e torna-o igual ao movimento criador.
SpecificExecuteOn	Chama *ExeSwapMovement* para trocar dois elementos do vector de sequência de tarefas de uma solução.

Tabela 6.20 *MvSeqSwap* - métodos específicos (nível concreto)

Também as classes para definição dos movimentos do tipo *shift* possuem apenas métodos, descritos na Tabela 6.21, para o nível abstracto, e na Tabela 6.22, para o nível concreto.

Método	Descrição
MvShift	Construtor da classe. Cria e inicializa os atributos.
Operator=	Torna iguais dois movimentos (atributos e avaliações).
ExeShiftMovement	Desloca um elemento de um vector de inteiros para uma nova posição, com base na informação de posições dos atributos.

Tabela 6.21 *MvShift* - métodos específicos (nível abstracto)

Método	Descrição
SeqMvShift	Construtor da classe.
CreateNew	Cria um novo movimento.
Copy	Cria um novo movimento e torna-o igual ao movimento criador.
SpecificExecuteOn	Chama *ExeShiftMovement* para deslocar um elemento do vector de sequência de tarefas de uma solução.

Tabela 6.22 *MvSeqShift* - métodos específicos (nível concreto)

Geradores de movimentos

São considerados três geradores de movimentos, correspondentes às três estruturas de vizinhança adoptadas. A Figura 6.8 apresenta a estrutura de classes para estes geradores.

Tal como no caso dos geradores de incrementos, também para os geradores de movimentos os atributos dos métodos das classes derivadas são omitidos, por motivos de clareza de apresentação e uma vez que todos constituem concretizações de métodos abstractos da classe base, na qual os atributos estão explicitados.

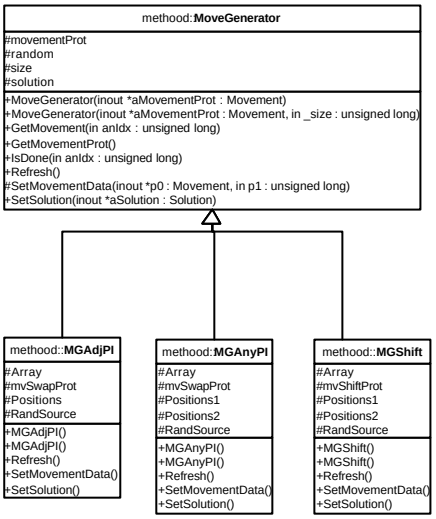


Figura 6.8 *Geradores de movimentos para problemas de sequenciamento* - diagrama de classes

A classe correspondente à vizinhança API é *MGAdjPI*. Os seus atributos e métodos específicos encontram-se na Tabela 6.23.

Atributo	**Descrição**
Array	Vector com sequência de tarefas da solução base da vizinhança.
MvSwapProt	Protótipo de movimentos do tipo *swap*.
Positions	Vector com sequência de posições gerada aleatoriamente, utilizado no caso de sub-vizinhanças. Uma tarefa será sempre trocada com a tarefa que a sucede na sequência.
RandSource	Gerador de números aleatórios.

Método	Descrição
MGAdjPI	Construtor da classe. No caso de ser definido com um tamanho, define uma sub-vizinhança. Sem tamanho, define uma vizinhança completa.
Refresh	Tratando-se de sub-vizinhança, gera sequência aleatória de posições.
SetMovementData	O parâmetro índice deste método corresponderá: ▪ no caso de sub-vizinhança, à posição dos vectores *Positions1* e *Positions 2* em que se vai obter as posições das tarefas a trocar; ▪ no caso de vizinhança completa, à posição do vector de sequência de tarefas que contém a tarefa a trocar com a subsequente.
SetSolution	Actualiza variáveis de acordo com a nova solução. Chama o mesmo método da classe base.

Tabela 6.23 *MGAdjPI* - atributos e métodos específicos

A classe correspondente à vizinhança *swap* é *MGAnyPI*, cujos atributos e métodos específicos são apresentados na Tabela 6.24.

Atributo	Descrição
Array	Vector com sequência de tarefas da solução base da vizinhança.
MvSwapProt	Protótipo de movimentos do tipo *swap*.
Positions1, Positions 2	Vectores com sequência de trocas de posições, gerada aleatoriamente, utilizados em sub-vizinhanças. Para cada tarefa em *Positions1*, a troca realiza-se com a tarefa na posição correspondente em *Positions2*.
RandSource	Gerador de números aleatórios.
Método	**Descrição**
MGAnyPI	Construtor da classe. No caso de ser definido com um tamanho, define uma sub-vizinhança. Sem tamanho, define uma vizinhança completa.
Refresh	Tratando-se de sub-vizinhança, gera sequência aleatória de trocas.
SetMovementData	O parâmetro índice deste método corresponderá: ▪ no caso de sub-vizinhança, à posição do vector *Positions* em que se vai obter a posição da tarefa a trocar com a tarefa seguinte; ▪ no caso de vizinhança completa, a um dos movimentos da vizinhança, de acordo com o mapeamento apresentado no Anexo C, que permite percorrer toda a vizinhança.
SetSolution	Actualiza variáveis de acordo com a nova solução. Chama o mesmo método da classe base.

Tabela 6.24 *MGAnyPI* - atributos e métodos específicos

A classe correspondente à vizinhança *shift* é *MGShift*. Na Tabela 6.25 são apresentados os seus atributos e métodos específicos.

Atributo	**Descrição**
Array	Vector com sequência de tarefas da solução base da vizinhança.
MvShiftProt	Protótipo de movimentos do tipo *shift*.
Positions1, *Positions 2*	Vectores com sequência de deslocamentos de posições, gerada aleatoriamente, utilizado no caso de sub-vizinhanças. As posições iniciais das tarefas a deslocar encontram-se em *Positions1*, e as finais nas posições correspondentes, em *Positions 2*.
RandSource	Gerador de números aleatórios.
Método	**Descrição**
MGShift	Construtor da classe. No caso de ser definido com um tamanho, define uma sub-vizinhança. Sem tamanho, define uma vizinhança completa.
Refresh	Tratando-se de sub-vizinhança, gera sequência aleatória de deslocamentos.
SetMovementData	O parâmetro índice deste método corresponderá: ▪ no caso de sub-vizinhança, à posição do vector *Positions* em que se vai obter a posição da tarefa a trocar com a tarefa seguinte; ▪ no caso de vizinhança completa, a um dos movimentos da vizinhança, de acordo com o mapeamento apresentado no Anexo C, que permite percorrer toda a vizinhança.
SetSolution	Actualiza variáveis de acordo com a nova solução. Chama o mesmo método da classe base.

Tabela 6.25 *MGShift* - atributos e métodos específicos

Avaliação de movimentos

É criado um avaliador de movimentos para cada possível combinação de um objectivo (soma ponderada dos atrasos, número de tarefas atrasadas e atraso máximo) com um tipo de movimento (*swap* e *shift*). Na Figura 6.9 é apresentada a estrutura de classes para estes avaliadores.

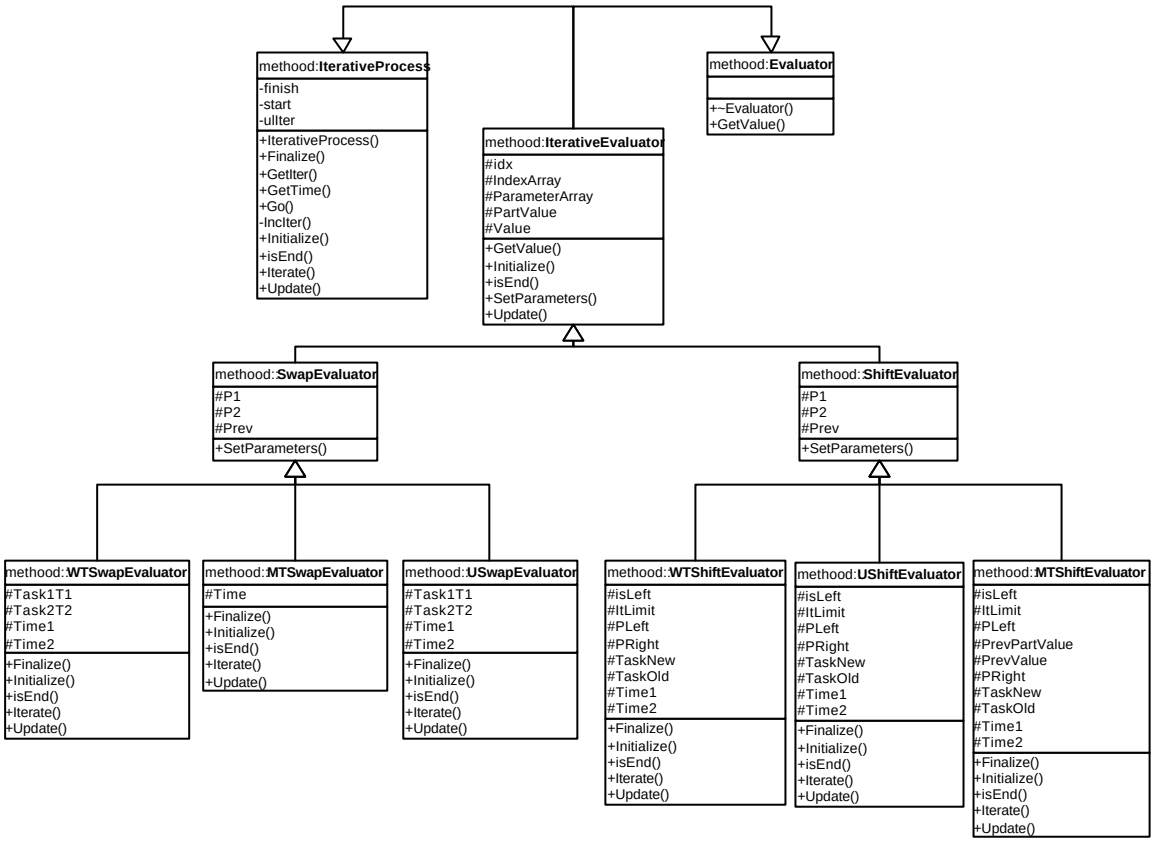


Figura 6.9 *Avaliação de movimentos para problemas de sequenciamento* - diagrama de classes

A hierarquia de herança aqui construída é semelhante à hierarquia apresentada para a avaliação de soluções, na parte relativa à utilização da classe *IterativeEvaluator*. É definido um nível intermédio de abstracção, com classes de avaliação relativas a cada um dos tipos de movimento, *SwapEvaluator* e *ShiftEvaluator*, que são depois especializadas para cada um dos objectivos.

Nesta última especialização seria possível usar herança múltipla, com a derivação adicional de *AdditiveEvaluator* e *MaxFilterEvaluator*, de forma idêntica ao realizado na avaliação de soluções. No entanto, enquanto a herança efectivamente considerada é essencial, uma vez que é ela que permite acoplar qualquer avaliador para movimentos de um tipo aos movimentos desse mesmo tipo, esta herança adicional não o é. Dado que a utilização de herança múltipla introduz alguma degradação no desempenho, evitou-se aqui a sua utilização,

dado tratar-se de uma parte do *framework* de utilização intensiva, na qual a eficiência de execução deverá constituir uma preocupação.

SwapEvaluator contém a informação específica para avaliação de um movimento do tipo *swap*. Os seus atributos e métodos específicos são apresentados na Tabela 6.26.

Atributo	**Descrição**
P1	Apontador para a posição da primeira tarefa a ser trocada.
P2	Apontador para a posição da segunda tarefa a ser trocada.
Prev	Apontador para o valor actual do objectivo a ser avaliado.
Método	**Descrição**
SetParameters	Inicializa atributos específicos; chama o mesmo método da classe base.

Tabela 6.26 *SwapEvaluator* - atributos e métodos específicos

ShiftEvaluator contém a informação específica para avaliação de um movimento do tipo *shift*. Os seus atributos e métodos específicos são apresentados na Tabela 6.27.

Atributo	**Descrição**
P1	Apontador para a posição da tarefa a deslocar.
P2	Apontador para a posição para a qual a tarefa deverá ser deslocada.
Prev	Apontador para o valor actual do objectivo a ser avaliado.
Método	**Descrição**
SetParameters	Inicializa atributos específicos; chama o mesmo método da classe base.

Tabela 6.27 *ShiftEvaluator* - atributos e métodos específicos

Ambos os tipos de movimento implicam alterações dos tempos de conclusão apenas no conjunto de tarefas entre as duas posições consideradas (para troca, no caso de *swap*, ou para deslocação, no caso de *shift*).

Nos casos do *número de tarefas atrasadas* e da *soma ponderada dos atrasos*, será suficiente acumular as variações do valor da função objectivo para cada tarefa pertencente ao referido conjunto, para se obter a variação total.

No caso do *atraso máximo*, a análise desse conjunto de tarefas poderá não ser suficiente para obter a variação pretendida. Se nesse conjunto de tarefas ocorrer um valor de atraso máximo superior ao actual, a variação será a diferença entre esses dois valores. Se tal não ocorrer, será também necessário avaliar os atrasos fora do conjunto de tarefas, e considerar o novo máximo global para a obtenção da variação pretendida.

A classe correspondente ao objectivo *soma ponderada dos atrasos*, para o movimento do tipo *Swap* é *WTSwapEvaluator*. Os seus atributos e métodos específicos são apresentados na Tabela 6.28.

Atributo	**Descrição**
Time1	Acumulador do tempo de processamento para a sequência actual.
Time2	Acumulador do tempo de processamento para a nova sequência.
Task1T1	Instante de conclusão da primeira tarefa na sequência anterior.
Task2T2	Instante de conclusão da segunda tarefa na nova sequência.
Método	**Descrição**
Initialize	Inicializa *Value*. Acumula o tempo de processamento das tarefas anteriores à primeira posição da troca, a partir do qual actualiza *Time1*, *Time2*, *Task1T1* e *Task1T2*. Coloca o índice na posição posterior à primeira posição da troca.
Iterate	Verifica, para uma tarefa, se houve alteração do seu atraso. Em caso positivo, coloca a variação do atraso ponderado em *PartValue*.
Update	Chama o mesmo método da classe base. Soma *PartValue* a *Value*.
IsEnd	Verdadeiro quando o índice atinge a segunda posição da troca.
Finalize	Calcula variações para tarefas das posições da troca; actualiza *Value*.

Tabela 6.28 *WTSwapEvaluator* - atributos e métodos específicos

USwapEvaluator disponibiliza a avaliação de movimentos do tipo *Swap* para o objectivo *número de tarefas atrasadas*. A implementação desta classe é muito semelhante à da anterior, pelo que se não procederá à sua descrição detalhada. Com efeito, a classe possui os mesmos atributos que a classe anterior e a implementação dos métodos apenas difere no facto de a determinação da variação apenas incidir sobre a alteração do estado de atraso de cada tarefa.

Ainda para movimentos do tipo *Swap*, a avaliação para o objectivo *atraso máximo* é implementada pela classe *MTSwapEvaluator*. Os seus atributos e métodos específicos são apresentados na Tabela 6.29.

Atributo	**Descrição**
Time	Tempo total de processamento acumulado até à iteração actual.
Método	**Descrição**
Initialize	Inicializa *Value*. Acumula o tempo de processamento das tarefas anteriores à primeira posição da troca. Actualiza *Value* com o atraso da segunda tarefa da troca, na sua nova posição. Coloca o índice na posição posterior à primeira posição da troca.
Iterate	Para uma tarefa, coloca em *PartValue* o valor do respectivo atraso.
Update	Incrementa o índice. Se *PartValue* for superior a *Value*, torna *Value* igual a *PartValue*.
IsEnd	Verdadeiro quando o índice atinge a segunda posição da troca.
Finalize	Calcula os atrasos para as tarefas das posições da troca e actualiza *Value*. Se *Value* for superior ao valor anterior *Prev*, a variação é dada pela diferença entre os dois valores. Caso contrário será necessário avaliar os atrasos também fora do conjunto de tarefas, e considerar o novo valor máximo global para a obtenção da variação pretendida.

Tabela 6.29 *MTSwapEvaluator* - atributos e métodos específicos

Relativamente à classe *MTShiftEvaluator*, esta implementa a avaliação do objectivo *atraso máximo* para movimentos do tipo *shift*. Os seus atributos e métodos específicos são apresentados na Tabela 6.30.

Atributo	**Descrição**
Time	Tempo total de processamento acumulado até à iteração actual.
isLeft	Verdadeira se o movimento for uma deslocação para a esquerda.
TaskNew	Novo tempo de conclusão da tarefa deslocada.
TaskOld	Anterior tempo de conclusão da tarefa deslocada.
Pleft	Posição mais à esquerda do bloco de tarefas que mudam de posição.
Pright	Posição mais à direita do bloco de tarefas que mudam de posição.
ItLimit	Posição até à qual será necessário iterar.
Método	**Descrição**
Initialize	Verifica se se trata de uma deslocação para a esquerda ou para a direita. Actualiza *isLeft*, *PLeft*, *PRight* e *ItLimit*. Acumula o tempo de processamento das tarefas anteriores a *PLeft*, a partir do qual actualiza *Time1*, *Time2*, *TaskNew* e *TaskOld*. Coloca o índice na posição mais à esquerda, se for uma deslocação para a esquerda, ou na posição posterior a essa, no caso contrário.
Iterate	Verifica, para uma tarefa, se houve alteração do seu atraso. Em caso positivo, coloca a variação do atraso ponderado em *PartValue*.
Update	Chama o mesmo método da classe base. Soma *PartValue* a *Value*.
isEnd	Verdadeiro quando o índice atinge *ItLimit*.
Finalize	Calcula, da forma adequada a cada um dos sentidos de deslocação, as variações para a tarefa deslocada e actualiza *Value*. Se *Value* for superior ao valor anterior *Prev*, a variação é dada pela diferença entre os dois valores. Caso contrário será necessário avaliar os atrasos também fora do conjunto de tarefas, e considerar o novo valor máximo global para a obtenção da variação pretendida.

Tabela 6.30 *MTShiftEvaluator* - atributos e métodos específicos

A classe correspondente ao objectivo *soma ponderada dos atrasos*, para o movimento do tipo *Shift*, é *WTShiftEvaluator*. Os seus atributos e métodos específicos são apresentados na Tabela 6.31.

Atributo	**Descrição**
Time1	Acumulador do tempo de processamento para a sequência actual.
Time2	Acumulador do tempo de processamento para a nova sequência.
IsLeft	Verdadeira se o movimento for uma deslocação para a esquerda.
TaskNew	Novo tempo de conclusão da tarefa deslocada.
TaskOld	Anterior tempo de conclusão da tarefa deslocada.
Pleft	Posição mais à esquerda do bloco de tarefas que mudam de posição.
Pright	Posição mais à direita do bloco de tarefas que mudam de posição.
ItLimit	Posição até à qual será necessário iterar.
Método	**Descrição**
Initialize	Verifica se se trata de uma deslocação para a esquerda ou para a direita. Actualiza *isLeft*, *PLeft*, *PRight* e *ItLimit*. Acumula o tempo de processamento das tarefas anteriores a *PLeft*, a partir do qual actualiza *Time1*, *Time2*, *TaskNew* e *TaskOld*. Coloca o índice na posição mais à esquerda, se for uma deslocação para a esquerda, ou na posição posterior a essa, no caso contrário.
Iterate	Verifica, para uma tarefa, se houve alteração do seu atraso. Em caso positivo, coloca a variação do atraso ponderado em *PartValue*.
Update	Chama o mesmo método da classe base. Soma *PartValue* a *Value*.
isEnd	Verdadeiro quando o índice atinge *ItLimit*.
Finalize	Calcula, da forma adequada a cada um dos sentidos de deslocação, as variações para a tarefa deslocada e actualiza *Value*.

Tabela 6.31 *WTShiftEvaluator* - atributos e métodos específicos

A descrição da classe *UShiftEvaluator*, que disponibiliza a avaliação do objectivo *número de tarefas atrasadas* para movimentos do tipo *shift*, não será aqui realizada, pelos motivos já enunciados a respeito da classe *USwapEvaluator*.

6.6 Conclusões

Os problemas de escalonamento possuem uma enorme importância prática, cobrindo uma grande variedade de situações práticas que, de forma genérica, podem ser descritas como de afectação de recursos limitados a tarefas. Em contextos práticos, os problemas de escalonamento possuem, frequentemente, uma natureza multicritério, que frequentemente não é tratada de forma adequada.

No âmbito da gestão de operações, os problemas de escalonamento de uma máquina assumem particular relevância, sendo de aplicação importante em situações de estrangulamento da produção ou em abordagens de decomposição de problemas complexos. São problemas interessantes, já que conjugam simplicidade de descrição e interesse (prático e teórico), com um grande número de variantes.

Muitas destas variantes são problemas NP-difíceis e, como tal, a aplicação de técnicas exactas e de regras de prioridade simples defronta-se com diversas dificuldades: no caso das primeiras, as fortes exigências computacionais em termos de tempo e memória; no caso das segundas, o facto de não produzirem, de forma consistente, boas soluções. A aplicação de meta-heurísticas permite exactamente ir ao encontro destas dificuldades.

No contexto desta dissertação, apresentam particular interesse as aplicações com um só objectivo, em especial de SA e TS, tendo-se verificado na literatura, a referência a um grande número de abordagens. Contudo, os componentes relacionados com o problema seguem, essencialmente, os mesmos esquemas nas várias abordagens: as soluções são representadas por permutações ou vectores binários; muitos algoritmos construtivos baseiam-se em regras de prioridade simples ou compostas; as estruturas de vizinhança utilizadas são *Adjacent Pairwise Interchange*, *Swap* e *Insert*, ou suas variantes. A configuração dos algoritmos é, porventura, o aspecto mais diferenciador entre as distintas abordagens.

Partindo da sistematização realizada sobre a aplicação de procedimentos baseados em pesquisa local, o *framework* desenvolvido foi aplicado ao problema em estudo - escalonamento multiobjectivo de tarefas numa máquina, minimizando o atraso máximo, o número de tarefas atrasadas e a soma ponderada dos atrasos.

Tal aplicação envolveu: a definição do esquema dos dados que definem uma instância do problema a resolver; a derivação das classes relativas à solução (representação baseada em permutações) e avaliação da solução (uma para cada um dos três objectivos); a derivação das classes relativas às estruturas de vizinhança: movimentos e atributos (*swap* e *shift*), geradores de movimentos (um para cada uma das três estruturas de vizinhança "clássicas") e avaliação dos movimentos (um para cada combinação movimento - objectivo); a derivação das classes relativas aos algoritmos construtivos: incrementos (apenas um tipo de incremento) e geradores de incrementos (geração aleatória, regra EDD, regra de Moore e regra ATC).

Desta aplicação conclui-se ser o *framework* adequado para a modelação dos componentes dependentes do problema: as classes abstractas, e relações entre estas, propostas para este efeito, permitem contemplar os requisitos de modelação de um problema multiobjectivo, com várias estruturas de vizinhança e vários algoritmos construtivos, e possuem uma correspondência efectiva com os componentes de meta-heurísticas, tal como encarados, de forma geral, na literatura.

A aplicação foi realizada com um esforço relativamente reduzido, na ordem de 1 pessoa-mês, tendo sido desenvolvidas cerca de 20 classes que correspondem a aproximadamente 35% do total de linhas de código (do conjunto do *framework* e da aplicação).

7
ESTUDO COMPUTACIONAL

Com o estudo computacional apresentado neste capítulo, pretendeu-se, de forma sucinta: estudar o comportamento das versões básicas de PSA e MOTS* no problema considerado; conduzir uma avaliação preliminar do interesse da utilização das estratégias de flexibilização propostas; avaliar o potencial do *framework* desenvolvido na sua aplicação à comparação de diversas configurações algorítmicas.

Na concepção do estudo computacional, e na apresentação e análise dos resultados, foram seguidas de perto as orientações de [Barr et al. 1995], para experiências computacionais na área dos métodos heurísticos.

Assim, a secção 7.1 consiste de uma descrição da concepção das experiências, contemplando os seguintes aspectos: objectivos do estudo, medidas de desempenho utilizadas, instâncias de teste, configurações algorítmicas a analisar, ambiente computacional, factores a analisar experimentalmente, estrutura das experiências e testes estatísticos para análise dos resultados. Nas secções 7.2 a 7.9 são apresentados e analisados os resultados, para os conjuntos de testes realizados sobre a versão básica de MOTS*, a versão básica de PSA, PSA com subvizinhanças, PSA com vizinhança variável, MOTS* com vizinhança variável, PSA com lista de candidatos, MOTS* com lista de candidatos e hibridização e paralelização de alto nível. Na secção 7.10 esses resultados são sintetizados e as conclusões mais relevantes são apresentadas. Um pequeno conjunto de comentários conclusivos encerra o capítulo.

7.1 Concepção das experiências

7.1.1 Objectivos

Os objectivos centrais prosseguidos pelo estudo computacional apresentado neste capítulo são os seguintes:

1. Comparar o desempenho de várias configurações da versão básica de PSA sobre um tipo particular de instâncias do problema de escalonamento em estudo. A caracterização do tipo de instâncias considerado neste trabalho é realizada em secção posterior deste capítulo.
2. Realizar idêntica comparação para a versão básica de MOTS*. A comparação de carácter mais geral entre PSA e MOTS*, que exigiria experiências de carácter distinto das realizadas com os objectivos restritos aqui enunciados, não é objecto deste estudo.
3. Sobre o mesmo tipo de instâncias, realizar uma primeira avaliação do potencial interesse da utilização de estratégias de listas de candidatos e vizinhanças variáveis, sobre as versões básicas de PSA e MOTS*, bem como da abordagem proposta de hibridização e paralelização de alto nível.
4. Avaliar, ainda que de forma indirecta e em termos gerais, o potencial do *framework* desenvolvido, como instrumento para comparação de diversas configurações algorítmicas.

7.1.2 Medidas de desempenho

Tendo em contexto os três primeiros objectivos estabelecidos, a avaliação dos algoritmos incide essencialmente sobre os seguintes aspectos do seu desempenho:

1. a qualidade das aproximações ao conjunto de soluções eficientes, que é, adiante, discutida em pormenor;
2. o esforço computacional, aferido pelo tempo de processador decorrido entre o início da execução do algoritmo e a última modificação da aproximação ao conjunto de soluções eficientes.

A avaliação, segundo estes dois aspectos, é realizada tendo em conta não só o desempenho médio dos algoritmos, mas também a robustez desse desempenho.

Dada a sua natureza, o quarto objectivo é, naturalmente, menos passível de uma apreciação quantitativa, sendo antes avaliado de uma forma global, com base no conjunto de toda a investigação e experiências realizadas.

7.1.3 Qualidade de aproximação ao conjunto de soluções não-dominadas

Existe, actualmente, alguma diversidade nas propostas de diferentes autores relativamente à avaliação de uma aproximação (ou *conjunto de aproximação*) *A* a um conjunto de soluções não-dominadas (ou *conjunto de referência*) *R*. Em [Hansen, Jaskiewicz 1998] são comentadas algumas medidas utilizadas na literatura, para o estudo de meta-heurísticas multiobjectivo:

- Medidas de *cardinalidade*, como o número (ou percentagem) de soluções de referência encontradas. Apresentam como inconveniente principal a insensibilidade a melhorias das aproximações, quer ao nível da proximidade ao conjunto de referência, quer ao nível da dispersão sobre esse conjunto.
- A medida de distância proposta em [Czyzac, Jaszkiewicz 1998],

$$D1_R(A,\ddot{e}) = \frac{1}{|R|}\sum_{r\in R}\min_{z\in A}\{d(r,z)\}, \tag{7.1}$$

em que

$$d(r,z) = \max_j\{l_j(r_j - z_j)\} \tag{7.2}$$

e $\ddot{e} = [l_1,...,l_J]$, $l_j = \frac{1}{Range_j}$, $j = 1,...,J$, sendo $Range_j$ a gama de valores do objectivo *j* no conjunto de referência *R*.

Esta medida pode ser interpretada como uma média das perdas máximas sobre todos os objectivos, ao tomar-se a aproximação *A* em vez do conjunto de referência *R*, com as perdas expressas em percentagem das gamas de valores dos objectivos, no conjunto *R*.

De acordo com [Hansen, Jaskiewicz 1998], esta medida é, de facto, uma média ponderada, na qual todas as soluções do conjunto de referência têm peso idêntico. Não existindo uma distribuição uniforme das soluções do conjunto de referência, os resultados obtidos com esta medida poderão ser contra-intuitivos. A verificação da uniformidade da distribuição das soluções do conjunto de referência deverá, portanto, preceder a utilização desta medida. No caso de a uniformidade se não verificar, poderá ser utilizado um conjunto de pesos, para

cada solução, que permita atribuir menor importância às soluções situadas em áreas de grande densidade.

- Uma outra medida, igualmente proposta em [Czyzac, Jaszkiewicz 1998],

$$D2_R(A,\ddot{e}) = \max_{r\in R}\left\{\min_{z\in A}\{d(r,z)\}\right\}, \tag{7.3}$$

ao trocar o valor médio pelo valor máximo das perdas, apresenta inconvenientes semelhantes aos das medidas de cardinalidade.

No presente estudo computacional utiliza-se, para avaliação da qualidade de aproximação, a medida de distância $D1$, sendo o conjunto de referência para cada instância do problema, formado pelas soluções não-dominadas do conjunto de todas as aproximações resultantes da execução da totalidade dos algoritmos a estudar.

No Anexo D apresenta-se um estudo sobre a uniformidade da distribuição das soluções nos conjuntos de referência obtidos neste trabalho, tendo-se concluído, com base nesse estudo, da adequação de $D1$ para a avaliação da qualidade de aproximação nas instâncias consideradas. O Anexo E apresenta a composição dos conjuntos de referência.

7.1.4 Instâncias de teste

Existem na literatura referências a vários conjuntos de instâncias de problemas de escalonamento de tarefas numa máquina, disponíveis para a realização de testes computacionais. Um desses conjuntos, referido com frequência recentemente, está disponibilizado, no âmbito da iniciativa *OR-Library* [Beasley 1990], em *http://mscmga.ms.ic.ac.uk/info.html*. Estas instâncias têm sido usadas em estudos do problema de minimização da soma ponderada dos atrasos, constituindo uma base natural para utilização no contexto do problema multiobjectivo aqui abordado.

O processo de geração das instâncias, descrito pormenorizadamente em [Crauwels et al. 1998], compreende, de forma sucinta, os seguintes passos:

- Geração, para cada tarefa, de um tempo de processamento, a partir de uma distribuição uniforme [1, 100], e de um peso, a partir de uma distribuição uniforme [1, 10] (ambos os valores inteiros).
- Definição de diversos graus de dificuldade das instâncias, correspondentes a distintas combinações de níveis da gama relativa de datas de entrega (RDD, de *"relative range of due dates"*) e do factor de atraso médio (TF, de *"average tardiness factor"*). Para ambos os factores, os níveis considerados estão no conjunto {0.2, 0.4, 0.6, 0.8, 1.0}, existindo, consequentemente, 25 pares de valores possíveis.

- Geração, para cada tarefa, de uma data de entrega, inteira, a partir de uma distribuição uniforme [P(1-TF-RDD/2), P(1-TF+RDD/2)], sendo P o somatório de todos os tempos de processamento.
- Para cada par de valores (RDD, TF), foram gerados cinco problemas, dando origem a 125 instâncias. O processo é realizado para dimensões do problema de 40, 50 e 100 tarefas.

Devido à carga computacional inerente ao estudo multiobjectivo aqui considerado, apenas foram consideradas as 10 primeiras instâncias do problema com 40 tarefas. As cinco primeiras correspondem ao par (RDD = 0.2 , TF = 0.2), enquanto as cinco seguintes correspondem ao par (RDD = 0.2, TF = 0.4). Verificou-se, após um extenso conjunto de testes, que os resultados obtidos com a primeira das 10 instâncias se situavam relativamente deslocados das gamas de resultados das restantes instâncias, pelo que foram excluídos das análises realizadas.

7.1.5 Configuração dos algoritmos

Opções gerais

Alguns aspectos de configuração dos algoritmos têm um carácter geral, relacionando-se de perto com o *template* definido para a pesquisa local multiobjectivo. Na Tabela 7.1 apresentam-se as opções de configuração relativas a esses aspectos.

Consideram-se dois valores de nível intermédio para o número de soluções da população. Estes valores são múltiplos do número de algoritmos construtivos (4), por forma a que exista um número idêntico de soluções iniciais construídas por cada algoritmo. A população inicial será, portanto, constituída por soluções de qualidade em cada um dos objectivos individualmente, e um conjunto de soluções aleatórias.

Dadas as características do problema, a muitas soluções distintas poderão corresponder vectores de critérios idênticos. Como exemplo, permutações das primeiras posições da sequência de tarefas, sem alteração das restantes, conduzirão a várias soluções distintas, sem que haja violação das datas de entrega e portanto sem que sejam alterados os valores dos vários objectivos. Para evitar a sobrecarga computacional associada ao processamento de soluções nesta situação, apenas são consideradas, para inserção no conjunto de aproximação, soluções com vectores de critérios distintos.

Opção	Configuração
Dimensão da população	Duas dimensões: 8 e 16 soluções
Soluções iniciais	Cada um dos 4 algoritmos construtivos contribui com um idêntico número de soluções iniciais
Inserção no conjunto de aproximação	Só permitida a soluções com vectores de critérios distintos
Critério de paragem	Realização de 1000 iterações sem alteração do conjunto de aproximação

Tabela 7.1 Opções de configuração gerais

O critério de paragem seleccionado consiste na realização de um número fixo de iterações, sem alteração do conjunto de aproximação. Este critério dota os algoritmos de alguma adaptabilidade relativamente ao esforço computacional a realizar, permitindo a continuação da pesquisa enquanto o algoritmo for melhorando a aproximação ao conjunto de soluções eficientes. O limiar estabelecido, de 1000 iterações, visa permitir um nível razoável de esforço de pesquisa para melhorar essa aproximação, sem colocar o esforço computacional correspondente em níveis incomportáveis.

Versão básica de PSA

Para o caso particular da versão básica de PSA, é necessário definir um conjunto de opções específicas, apresentadas na Tabela 7.2.

Para o factor de arrefecimento utiliza-se um valor, relativamente elevado, de 0.95 ([Almeida, Centeno 1998], [Viana 1997]). O factor de actualização dos pesos toma o valor 1.1, próximo da unidade, de acordo com [Czyzac, Jaszkiewicz 1998].

Para o comprimento do *plateau* consideram-se valores de 5, 50 e 200 iterações, por forma a permitir velocidades de arrefecimento razoavelmente distintas: nas 1000 iterações definidas para o critério de paragem, os comprimentos de *plateau* considerados permitirão realizar, respectivamente, 200, 20 e 5 descidas de temperatura.

Relativamente às funções de probabilidade, [Ulungu et al. 1997] referem que a sua escolha poderá ter alguma influência sobre o desempenho do algoritmo, embora esse efeito seja reduzido devido ao carácter estocástico do método. Arbitra-se, portanto, a utilização de uma das funções apresentadas por estes autores: o mínimo das probabilidades calculadas, com ponderação, separadamente em cada objectivo.

Opção	Configuração
Factor de arrefecimento	0.95
Factor de actualização de pesos	1.1
Plateau	5, 50, 200
Probabilidade de transição	Mínimo das probabilidades individuais em cada objectivo
Temperatura	2, 0.5, 0.01, 0.005
Vizinhança	Sub-vizinhança do tipo *Insert* com dimensão 1

Tabela 7.2 Opções de configuração específicas para a versão básica de PSA

Assim sendo, o conjunto de pesos para o qual esta probabilidade é mínima é o caso extremo em que o peso no objectivo com degradação máxima é unitário e os restantes nulos. É, portanto, esta a situação limite para a qual se avaliam as probabilidades iniciais.

São considerados quatro níveis de temperaturas iniciais, definidos tomando como referência um vector de critérios para o qual a degradação mais elevada, relativamente ao vector de critérios da solução actual, em qualquer dos objectivos, equivale a 5% das gamas equalizadas. Com os valores seleccionados procura-se realizar alguma cobertura do espectro das probabilidades de aceitação.

A probabilidade é, então, dada pela expressão

$$\min\left\{1, e^{-\frac{l \cdot d \cdot R_{eq}}{T}}\right\},$$

em que l é o peso (1), d a percentagem de degradação (5%), R_{eq} a gama equalizada e T a temperatura.

Consideram-se aproximações iniciais ao conjunto de soluções eficientes, determinadas a partir das soluções iniciais obtidas não aleatoriamente (Tabela 7.3). Nas dez instâncias testadas verificou-se que a aproximação era constituída pelas três soluções consideradas. Para cada instância calculam-se as gamas, os factores de equalização de gamas e a gama equalizada (Tabela 7.4), valor a partir do qual é possível avaliar a probabilidade de aceitação, para diversos níveis da temperatura T (Tabela 7.5).

A vizinhança utilizada é do tipo *Insert*, uma vez que, de acordo com [Laguna, Glover 1993], de entre as vizinhanças simples, esta é a que tem proporcionado, com alguma consistência, os melhores resultados em problemas de escalonamento. A dimensão da subvizinhança é unitária, segundo a configuração clássica de SA.

Instância	EDD			Moore			ATC			Gamas		
	ΣU_j	L_{max}	$\Sigma w_j T_j$	ΣU_j	L_{max}	$\Sigma w_j T_j$	ΣU_j	L_{max}	$\Sigma w_j T_j$	ΣU_j	L_{max}	$\Sigma w_j T_j$
1	3	581	5444	6	210	1588	6	408	1062	3	371	4382
2	3	591	4977	10	213	5226	6	352	1465	7	378	3761
3	2	402	2658	8	187	3051	3	288	951	6	215	2100
4	3	353	5551	10	206	5527	6	393	2252	7	187	3299
5	3	412	3520	5	218	4030	5	352	1433	2	194	2597
6	7	804	20966	14	591	23691	12	711	7587	7	213	16104
7	7	957	20453	17	611	33547	9	884	6673	10	346	26874
8	7	603	18254	18	560	23032	11	876	8608	11	316	14424
9	8	786	30507	19	685	34596	11	892	17256	11	207	17340
10	8	918	29129	15	705	44178	10	1131	10280	7	426	33898

Tabela 7.3 Aproximações iniciais aos conjuntos de soluções eficientes

Instância	Gamas			Factores de equalização			Gamas equalizadas
	ΣU_j	L_{max}	$\Sigma w_j T_j$	ΣU_j	L_{max}	$\Sigma w_j T_j$	
1	3	371	4382	1.12E-01	9.06E-04	7.67E-05	0.336257
2	7	378	3761	2.08E-02	3.86E-04	3.88E-05	0.145769
3	6	215	2100	2.86E-02	7.99E-04	8.18E-05	0.171794
4	7	187	3299	2.12E-02	7.94E-04	4.50E-05	0.148508
5	2	194	2597	2.53E-01	2.61E-03	1.95E-04	0.505540
6	7	213	16104	2.11E-02	6.93E-04	9.17E-06	0.147614
7	10	346	26874	1.03E-02	2.97E-04	3.83E-06	0.102927
8	11	316	14424	8.56E-03	2.98E-04	6.53E-06	0.094143
9	11	207	17340	8.71E-03	4.63E-04	5.52E-06	0.095798
10	7	426	33898	2.07E-02	3.41E-04	4.28E-06	0.145234

Tabela 7.4 Gamas, factores de equalização e gamas equalizadas

Temperatura inicial	Probabilidade de aceitação (%)		
	Média	Mínima	Máxima
2	99.53	98.74	99.76
0.5	98.13	95.07	99.06
0.01	44.51	7.98	62.46
0.005	22.73	0.64	39.01

Tabela 7.5 Probabilidade de aceitação inicial, em função da temperatura, nas instâncias consideradas

Versão básica de MOTS*

Também para o caso da versão básica de MOTS*, existem diversas opções específicas a realizar, sistematizadas na Tabela 7.6.

Para o comprimento da lista tabu são utilizados os valores 0, 7 e 15. Um comprimento nulo permite avaliar o efeito da (não) presença da lista tabu. 7 é um dos valores frequentemente utilizados em abordagens a problemas de escalonamento de tarefas numa máquina ([Madureira, Sousa 1996], [James 1997], [James, Buchanan 1997, 1998], [Almeida, Centeno 1998], [Crawels et al. 1998]). Para valores superiores a 7, é referida a utilização de um comprimento de 15 em [James, Buchanan 1997, 1998] e [James 1997].

O intervalo para *drift* é de 100 iterações, valor igualmente referido na literatura [Viana 1997].

Opção	Configuração
Comprimento da lista tabu	0, 7, 15
Intervalo de iterações para *drift*	100
Vizinhança	Sub-vizinhança do tipo *Insert* com dimensões relacionadas com o tamanho das instâncias: $n/4$, $n/2$ e n.

Tabela 7.6 Opções de configuração específicas para a versão básica de MOTS*

A vizinhança utilizada é do tipo *Insert*, pelos motivos já enunciados a respeito da configuração da versão básica de PSA. São utilizadas sub-vizinhanças de pequenas dimensões, sendo a dimensão máxima igual ao número de tarefas *n.* Esta é a sub-vizinhança de dimensão mínima com probabilidade não nula de conter deslocamentos de todas as tarefas. As dimensões inferiores utilizadas são $n/4$ e $n/2$.

Vizinhanças variáveis

As vizinhanças variáveis são aplicadas apenas às configurações básicas de PSA e MOTS* que apresentam o melhor e o pior desempenhos médios, ao nível da qualidade de aproximação. Considera-se a utilização de uma vizinhança variável, com a sequência [*Swap*, *Insert*]. Nesta sequência há um impacto crescente na manipulação de uma solução, uma vez que *Swap* apenas troca de posição qualquer par de tarefas, enquanto *Insert* não só movimenta uma tarefa para qualquer posição, como desloca de uma posição o conjunto de tarefas entre as posições inicial e final da tarefa. A dimensão das sub-vizinhanças utilizadas é, no caso do PSA,

unitária, enquanto na MOTS* corresponde à dimensão da correspondente configuração básica.

Estratégias de listas de candidatos

Também a estratégia de lista de candidatos *elite* é aplicada apenas às configurações básicas de PSA e MOTS* que apresentam o melhor e o pior desempenhos médios, ao nível da qualidade das aproximações. As dimensões de lista a avaliar são 2, 5 e 10. A utilização de listas de candidatos com PSA implica naturalmente a consideração de sub-vizinhanças de dimensão superior à unitária. O leque de valores a utilizar para a dimensão da sub-vizinhança é idêntico ao das dimensões da lista.

Paralelização e hibridização de alto nível

As combinações a avaliar serão compostas de pares de algoritmos, um baseado em PSA e o outro em MOTS*, que na execução individual apresentem idênticos níveis de qualidade de aproximação. Serão seleccionados os pares com melhor e pior qualidade de aproximação média.

7.1.6 Ambiente computacional

A plataforma computacional utilizada para os testes tem a configuração apresentada na Tabela 7.7.

Marca	Compaq
Modelo	Armada 1750
Processador	Pentium II 333Mhz
RAM	64.0 MB
Disco	6.01 GB
Sistema operativo	Windows 98

Tabela 7.7 Configuração do computador pessoal utilizado nos testes computacionais

A aplicação foi desenvolvida em *Microsoft Visual C++ 6.0* (MSVC++ 6.0) *Enterprise Edition*, como *Win32 Console Application*. A única intervenção sobre a configuração do compilador localiza-se ao nível da optimização, orientada para a maximização da velocidade de execução do *software.*

As bibliotecas utilizadas são a STL, disponibilizada com o MSVC++ 6.0, e as bibliotecas da LEDA R4.1 para o MSVC++ 6.0, disponibilizadas em *www.mpi-sb.mpg.de/LEDA*. Para as configurações envolvendo paralelização, é utilizada a biblioteca *multi-threaded* de *run-time*. Para as restantes configurações, é utilizada a biblioteca *single-threaded*.

7.1.7 Factores a analisar experimentalmente

A realização do conjunto de testes computacionais propostos visa permitir analisar a influência de diversos factores variáveis das configurações dos algoritmos sobre o seu desempenho. Este desempenho é, conforme já referido, avaliado em duas perspectivas: a qualidade de aproximação e o tempo de execução.

Um outro factor relevante para o estudo do desempenho dos algoritmos, embora raramente tratado de uma maneira mais formal, é a própria instância a resolver (e cujas características poderão influenciar fortemente esse desempenho).

Factor Instância

As instâncias a testar deverão apresentar, de uma forma geral, níveis de dificuldade de resolução distintos, influenciando, assim, a qualidade das aproximações obtidas, ao nível dos respectivos valores médios, e igualmente da sua dispersão.

Também o tempo de execução será naturalmente influenciado pela instância, com destaque para dois aspectos em particular: a dimensão do conjunto de aproximação deverá ter uma influência determinante no tempo de execução, uma vez que, para cada solução pesquisada é necessário verificar se o respectivo vector de critérios é ou não dominado pelos vectores de critérios de todos os elementos do conjunto de aproximação; por outro lado, as características de cada instância impõem padrões de convergência específicos, que conduzirão a diferenças na relação qualidade / tempo de execução. Esta relação poderá ainda ser influenciada pelo critério de paragem adoptado, com paragem do algoritmo para taxas de convergência inferiores ao limite definido por aquele critério.

Relativamente à dispersão dos tempos de execução, será de esperar que esta apresente um crescimento com os respectivos valores médios, em virtude da presença de processos aleatórios nos algoritmos.

No âmbito do presente trabalho não serão investigados outros tipos de características das instâncias, na sua relação com o desempenho dos algoritmos.

Factor População

Para os dois algoritmos básicos em estudo, PSA e MOTS*, um dos parâmetros a variar é a dimensão da população. A um aumento da dimensão da população de soluções, corresponderá um aumento do esforço de pesquisa em cada iteração e, através das estratégias de definição de pesos, uma maior diversificação da pesquisa. O tempo de execução deverá, portanto, registar um incremento dos valores médios e, consequentemente, da dispersão, com o aumento da dimensão da população. Por seu lado, a qualidade média das aproximações deverá registar melhorias, juntamente com uma redução da dispersão.

São testados apenas dois valores para a dimensão da população que, por comparação com valores utilizados na literatura, se podem considerar de nível intermédio. Não se deverá, portanto, estar em presença de situações extremas, como a existência de um número excessivamente reduzido de soluções, não cobrindo razoavelmente a fronteira não-dominada, ou a existência de demasiadas soluções, obrigando a um excessivo esforço computacional e retardando a convergência para a fronteira não-dominada.

Previsivelmente, as melhorias na qualidade das aproximações, associadas ao aumento da dimensão da população, dependerão também das instâncias, uma vez que uma dada dimensão da população poderá ser mais adequada para instâncias com determinadas características, e menos para outras. Também os tempos de execução apresentarão naturalmente uma dependência em relação às instâncias, devendo registar-se incrementos mais acentuados, ao passar de uma população de 8 soluções para uma população de 16 soluções, nas instâncias com tempos de execução médios à partida também mais elevados.

Factores específicos

Em cada algoritmo, dois parâmetros específicos são objecto de "afinação": no caso da MOTS*, o tamanho da lista tabu e da subvizinhança, e no caso do PSA, a temperatura inicial e o comprimento do *plateau*.

Os parâmetros específicos deverão determinar diferentes níveis de intensificação, diversificação e aleatorização da pesquisa, com impactos distintos sobre a qualidade das aproximações obtidas, o tempo de execução, e a própria relação entre ambos. O estudo a realizar deverá permitir uma caracterização preliminar destes efeitos. Embora a realização *a priori* desta caracterização apresente algumas dificuldades, será útil esboçar um certo número de hipóteses gerais, que se referem de seguida.

No caso da MOTS*, o tempo de execução deverá crescer com a dimensão da lista tabu e da subvizinhança. Relativamente à qualidade das aproximações, esta será fortemente

determinada pelo compromisso entre a dimensão da subvizinhança que, se reduzida, induzirá uma diversificação muito elevada, e a dimensão da lista tabu, cujo crescimento conduz a uma maior intensificação da pesquisa.

No caso do PSA, partir de uma temperatura baixa ou atingi-la muito rapidamente, poderá conduzir a uma concentração prematura da pesquisa em soluções não-dominadas locais, enquanto o inverso poderá impedir uma estabilização mínima da pesquisa em áreas promissoras. O tempo de execução será fundamentalmente determinado pela taxa de convergência resultante deste compromisso, e a sua relação com o limite estabelecido pelo critério de paragem.

Uma vez que compromissos distintos entre os valores dos factores variáveis se deverão ajustar de forma distinta aos espaços de soluções específicos a cada instância, e às topologias neles criadas pelas estruturas de vizinhança consideradas, será de supor a existência de interacção entre os parâmetros específicos e as instâncias individuais.

Na medida em que as estratégias de definição dos pesos visam sobretudo definir direcções de pesquisa, desde que se não verifiquem as situações extremas descritas a respeito do factor população, a interacção com a dimensão da população deverá envolver apenas uma aproximação dos valores médios da qualidade obtida com os diferentes compromissos, para maiores dimensões.

Relativamente à dispersão dos resultados obtidos, deverão manter-se as relações enunciadas anteriormente: para os tempos de execução deverá manter-se o crescimento com os valores médios, enquanto para a qualidade de aproximação, as menores dispersões deverão acompanhar as melhores aproximações.

Factor abordagem de flexibilização

A utilização das abordagens de flexibilização propostas visa potenciar uma melhoria no desempenho das abordagens base, procurando melhorar tanto o compromisso qualidade / tempo de execução, como a robustez das abordagens.

A caracterização da interacção entre as configurações dos algoritmos de base, as configurações dos elementos de flexibilização e o próprio problema e tipo de instâncias consideradas apresenta algumas dificuldades, podendo este estudo computacional contribuir para a formulação de hipóteses sobre essa interacção. Neste contexto, não se avança, à partida, qualquer hipótese relativamente ao impacto da utilização destas abordagens.

7.1.8 Estrutura das experiências e análise estatística

A execução de cada configuração algorítmica apresentada em 7.1.5, sobre cada uma das 9 instâncias consideradas, é objecto de 10 repetições. Para cada execução determina-se a distância D1, entre a aproximação obtida e o conjunto de soluções não-dominadas para a instância correspondente, conjunto este construído a partir de todas as aproximações obtidas para a instância em causa. É ainda registado, para cada execução, o tempo correspondente ao último instante em que o conjunto de aproximação é actualizado.

A análise dos resultados envolve a utilização de um conjunto de métodos estatísticos, cujos princípios gerais, e aplicações concretas no presente estudo computacional, são apresentados de seguida:

- *Análise de variância (ANOVA).* O objectivo geral da análise de variância é testar a existência de diferenças significativas entre valores esperados de populações. Tal é conseguido através de uma análise da variância do conjunto de dados, decompondo-a no componente devido ao erro aleatório e nos componentes devidos às diferenças de médias. No caso de algum destes últimos componentes ser significativo, rejeita-se a hipótese nula da inexistência de diferenças significativas entre valores esperados, e aceita-se a hipótese alternativa de que os valores esperados são diferentes.

  A análise de variância é utilizada, no contexto deste estudo, para verificar se existem diferenças significativas de desempenho (qualidade de aproximação e tempo de execução) entre as diversas configurações algorítmicas, e averiguar que factores são significativos na explicação dessas diferenças de desempenho.

- *Teste de Tukey para a diferença de valores esperados*. O teste de Tukey, utilizado conjuntamente com a análise de variância, quando a hipótese de igualdade de valores esperados é rejeitada, permite verificar quais os que diferem significativamente, através de comparações de pares.

  Neste estudo, em particular, a aplicação deste teste permitirá verificar o conjunto das configurações algorítmicas com melhor desempenho (qualidade de aproximação e tempo de execução), ou seja, o conjunto das configurações algorítmicas com melhores resultados, e sem diferenças significativas entre eles.

- *Teste de Bartlett*. O teste de Bartlett tem como objectivo verificar a condição de homogeneidade da variância, que constitui um dos pressupostos da análise de variância.

- *Teste de Kolmogorov-Smirnov Lilliefors (K-S Lilliefors).* Este teste de qualidade de ajuste destina-se a avaliar a aderência entre uma distribuição de frequências associada a uma amostra constituída por observações quantitativas e uma distribuição Normal, com parâmetros estimados a partir da amostra.

  A aplicação do teste de K-S Lilliefors permite verificar pressupostos de normalidade de populações, para a realização da análise de variância.

- *Teste de correlação ordinal de Spearman.* Este teste, que visa verificar se duas variáveis estão ou não associadas, tem múltiplas aplicações ao longo deste estudo computacional como, por exemplo, a verificação da associação entre valores esperados e variâncias, ou da associação entre qualidade de aproximação e tempo de execução.

Todos os testes são realizados ao nível de significância de 5%. No caso dos estudos relativos às abordagens de flexibilização, dado o seu carácter preliminar, apenas se aplicam a análise de variância e o teste de *Tukey*.

Para apoiar a apresentação de resultados são utilizados vários tipos de diagramas:

- *Diagrama de médias.*

  Apresenta os valores médios da variável em observação para as várias classes consideradas.

- *Diagrama do tipo caixa ("box plot").*

  Permite sintetizar graficamente, para as várias classes consideradas, valores relativos à localização e à dispersão da variável em observação.

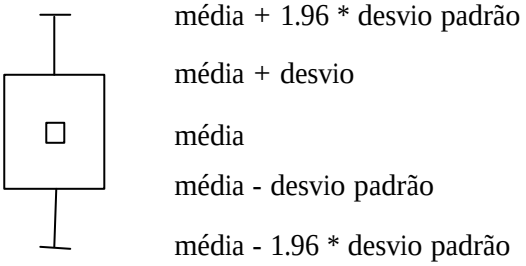


Figura 7.1 Diagrama do tipo caixa

- *Diagrama (x, y).*

  Representação dos dados de dois atributos num sistema de eixos ortogonais. Neste estudo, este tipo de diagrama é utilizado para representar as seguintes relações: médias e desvios padrão; qualidade de aproximação média e tempo de

execução médio; tempo de execução médio e número de soluções não dominadas.

A ferramenta informática utilizada para a análise estatística é o pacote *Statistica for Windows, Release 5.1D*, da *Statsoft, Inc.* (1996).

7.1.9 Notação

Resume-se na Tabela 7.8 a notação relativa aos factores, utilizada nos diversos diagramas. As notações específicas para os vários algoritmos são apresentadas na Tabela 7.10, enquanto na Tabela 7.9 é introduzida a notação e as unidades de medida relativas às variáveis.

Factor	Designação	Notação	Valores
Instância	*Instância*	INST	WT040002 - WT040010
Nº de soluções da população	*População*	POP	8, 16
Comprimento da lista tabu	*Lista*	LIST	0, 7, 15
Dimensão da subvizinhança	*Subvizinhança*	SUBVIZ	5, 10, 20, 40
Comprimento do *Plateau*	*Plateau*	PLATEAU	5, 50, 200
Temperatura inicial	*Temperatura*	TEMP	0.005, 0.01, 0.5, 2
Lista de candidatos	*Lista*	LISTC	2, 5, 10
Vizinhança variável	*Vizinhança*	VIZIN	(N)ormal, (V)ariável
Configuração algorítmica	*Configuração*	CONFIG	A/B - melhor/pior aproximação TS_A/TS_B - melhor/pior aproximação de MOTS* SA_A/SB_B - melhor/pior aproximação de PSA HP_A/HP_B - algoritmo híbrido paralelo de alto nível, com configurações base com melhor/pior qualidade de aproximação média

Tabela 7.8 Notação relativa aos factores

Variável	Notação	Unidade
D1	D1	-
Tempo de execução	T	segundo

Tabela 7.9 Notação relativa às variáveis

Algoritmo	Notação	
	Texto	**Diagramas**
Versão básica de MOTS*	(População, Lista, Subvizinhança)	População_Lista_Subvizinhança
Versão básica de PSA	(População, Temperatura, *Plateau*)	População_Temperatura_*Plateau*
PSA com subvizinhanças	(Configuração base, Subvizinhança)	ConfiguraçãoBase_Subvizinhança
PSA com lista de candidatos	(Configuração base, Lista de candidatos)	ConfiguraçãoBase_ListaDeCandidatos
MOTS* com lista de candidatos	(Configuração base, Lista de candidatos)	ConfiguraçãoBase_ListaDeCandidatos
PSA com vizinhança variável	(Configuração base, Tipo de vizinhança)	ConfiguraçãoBase_TipoDeVizinhança
MOTS* com vizinhança variável	(Configuração base, Tipo de vizinhança)	ConfiguraçãoBase_TipoDeVizinhança
Hibridização e paralelização de alto nível	Configuração	Configuração

Tabela 7.10 Notação relativa aos algoritmos

7.1.10 Verificação dos pressupostos das análises de variância

A condição de normalidade da distribuição dos erros é a menos crítica das condições de aplicação da análise de variância, que é pouco sensível, no conjunto, à não normalidade. Por outro lado, a condição de homogeneidade da variância tem uma importância relativamente secundária quando as amostras são todas do mesmo tamanho, o que sucede nos estudos aqui conduzidos.

No entanto, com o objectivo de verificar se em algum dos testes haveria desvios críticos destas condições, procedeu-se à realização de um conjunto de testes para verificação dos pressupostos das análises de variância, sendo apresentados na Tabela 7.11 os respectivos valores das estatísticas de teste e valores de prova. Para as duas variáveis consideradas, realizaram-se testes de K-S Lilliefors para a normalidade da distribuição dos erros, e testes de Bartlett para a homogeneidade da variância. Apenas para o tempo de execução em PSA com vizinhança variável, o valor de prova no teste de K-S Lilliefors excede o nível de 1%, garantido em todos os restantes testes. No entanto, como o valor é inferior a 10%, considera-se que também aqui o pressuposto é verificado.

Abordagem	D1				T			
	K-S Lilliefors		Bartlett		K-S Lilliefors		Bartlett	
	d	Valor de prova	c	Valor de prova	d	Valor de prova	c	Valor de prova
Versão básica de MOTS*	0.091	<1%	1297.4	0	0.140	<1%	2610.4	0
Versão básica de PSA	0.091	<1%	1412.2	0	0.096	<1%	1604.2	0
PSA com subvizinhanças	0.172	<1%	873.7	0	0.091	<1%	571.7	0
PSA com lista de candidatos	0.069	<1%	300.2	0	0.151	<1%	987.1	0
MOTS* com lista de candidatos	0.079	<1%	390.9	0	0.108	<1%	597.5	0
PSA com vizinhança variável	0.184	<1%	352.2	0	0.045	<10%	111.1	0
MOTS* com vizinhança variável	0.132	<1%	380.6	0	0.130	<1%	319.3	0
Hibridização e paralelização de alto nível								
Config. base com melhor qual. de aprox.	0.086	<1%	153.9	0	0.123	<1%	250.2	0
Config. base com pior qual.de aprox.	0.072	<1%	134.2	0	0.070	<1%	134.7	0

Tabela 7.11 Verificação dos pressupostos das análises de variâncias

7.2 Versão básica de MOTS*

7.2.1 Qualidade de aproximação

A tabela ANOVA para os valores de D1 obtidos com as várias configurações da versão básica de MOTS* é apresentada na Tabela 7.12.

Fonte	GL Fonte	DQM Fonte	GL Erro	DQM Erro	F	Valor de prova (%)
Instância	8	0.0729	1944	0.0006	116.6607	**0.00**
População	1	0.1282	8	0.0021	60.2260	**0.01**
Lista	2	0.0510	16	0.0052	9.8625	**0.16**
Subvizinhança	3	0.2040	24	0.0147	13.8390	**0.00**
Instância * População	8	0.0021	1944	0.0006	3.4043	**0.07**
Instância * Lista	16	0.0052	1944	0.0006	8.2641	**0.00**
População * Lista	2	0.0022	16	0.0006	3.4194	5.80
Instância * Subvizinhança	24	0.0147	1944	0.0006	23.5707	**0.00**
População * Subvizinhança	3	0.0001	24	0.0012	0.0580	98.12
Lista * Subvizinhança	6	0.1328	48	0.0058	22.9493	**0.00**
Instância * População * Lista	16	0.0006	1944	0.0006	1.0194	43.22
Instância * População * Subvizinhança	24	0.0012	1944	0.0006	1.9344	**0.43**
Instância * Lista * Subvizinhança	48	0.0058	1944	0.0006	9.2585	**0.00**
População * Lista * Subvizinhança	6	0.0009	48	0.0012	0.7864	58.49
Instância * População * Lista * Subvizinhança	48	0.0012	1944	0.0006	1.9068	**0.02**

Tabela 7.12 MOTS* - Tabela ANOVA para D1

Factor Instância

A existência de uma interacção de nível 4 significativa indica, em particular, que a interacção *População* * *Lista* * *Subvizinhança* se altera consoante a *Instância*. Graficamente (Figura 7.2), observa-se que o componente mais saliente da interacção é a **variação, conforme a instância, da proximidade entre os valores médios de D1** obtidos com as várias configurações. **A importância deste componente é igualmente observável nas restantes interacções com o factor *Instância***, que se apresentam também significativas.

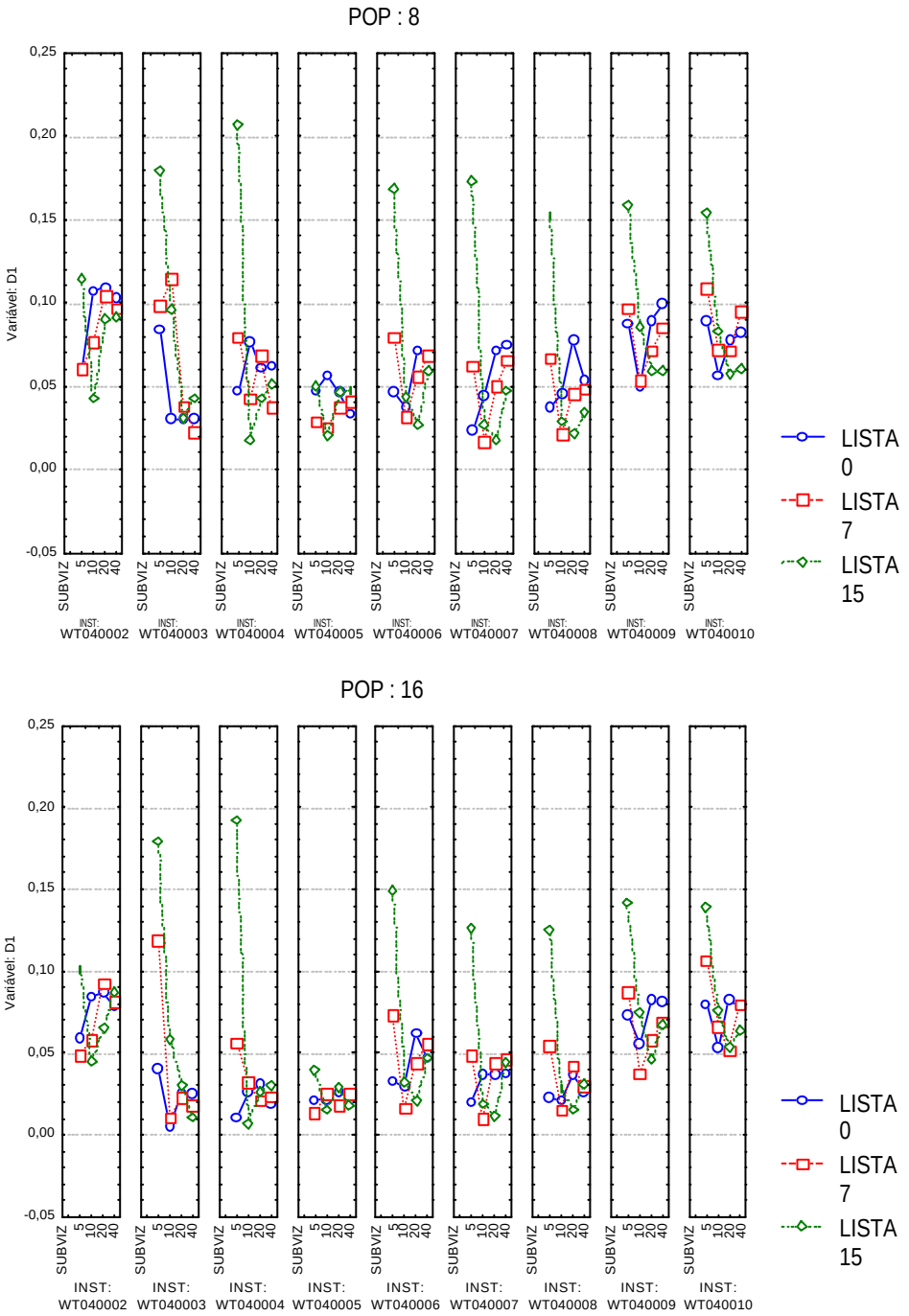


Figura 7.2 MOTS* - Diagramas de médias de D1 para *População *Instância * Lista * Subvizinhança*

Factor População

A menos de flutuações aleatórias, **o aumento da dimensão da *População* conduz a uma melhoria da qualidade das aproximações, mas sem alteração significativa da interacção *Lista * Subvizinhança***, conforme indicado pelo carácter não significativo da interacção que envolve os três factores. O diagrama de médias da Figura 7.3 permite ilustrar esta mesma observação. Embora graficamente se possa observar alguma aproximação dos valores médios, induzida pelo aumento da população, este efeito não é, contudo, estatisticamente significativo.

No diagrama do tipo caixa da Figura 7.4 é observável um **ligeiro decréscimo da dispersão dos resultados obtidos com as configurações englobando *Populações* de 16 soluções**.

Do carácter significativo da interacção *Instância * População* é observável que **a diferença entre os resultados para as duas dimensões de *População* varia conforme a *Instância***. Uma análise *post hoc*, baseada nos testes de *Tukey* para a igualdade de valor esperado, permitiu verificar que existem diferenças significativas entre os resultados obtidos para as dimensões de *População* consideradas para todas as *Instâncias*, à excepção das 9 e 10. Conforme já referido, a investigação desta interacção, que envolveria o aprofundamento do estudo das características das *Instâncias*, está fora do âmbito deste trabalho.

Interacção Lista * Subvizinhança

A interacção *Lista * Subvizinhança* é significativa, não se apresentando um claro significado individual para qualquer dos efeitos, conforme se pode observar na Figura 7.3. É, no entanto, de assinalar a significativa deterioração dos resultados proporcionados com dimensões da *Lista* de 7 e 15, para uma *Subvizinhança* de dimensão 5. Na origem deste efeito deverá estar um **excesso de diversificação, associado a uma *Subvizinhança* de dimensão reduzida, no âmbito do qual a intensificação conduzida pela *Lista* deverá condicionar a permanência na solução de atributos limitativos da melhoria da respectiva qualidade**. Para as restantes dimensões da *Subvizinhança*, observa-se a **importância da *Lista* para a intensificação da pesquisa, de forma a permitir um adequado compromisso com a diversificação introduzida pela *Subvizinhança***.

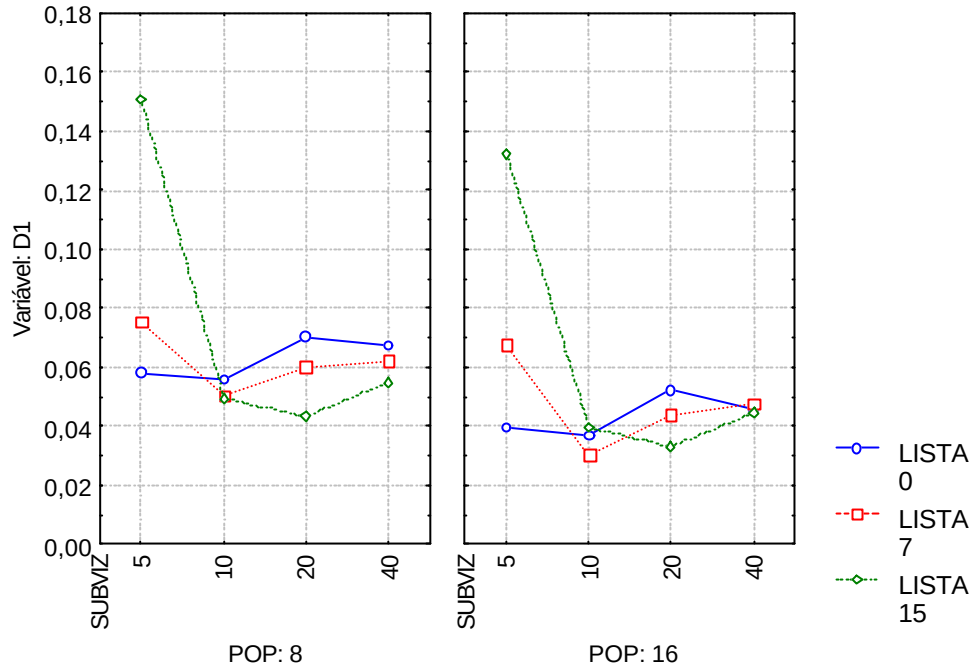


Figura 7.3 MOTS* - Diagramas de médias de D1 para *População * Lista * Subvizinhança*

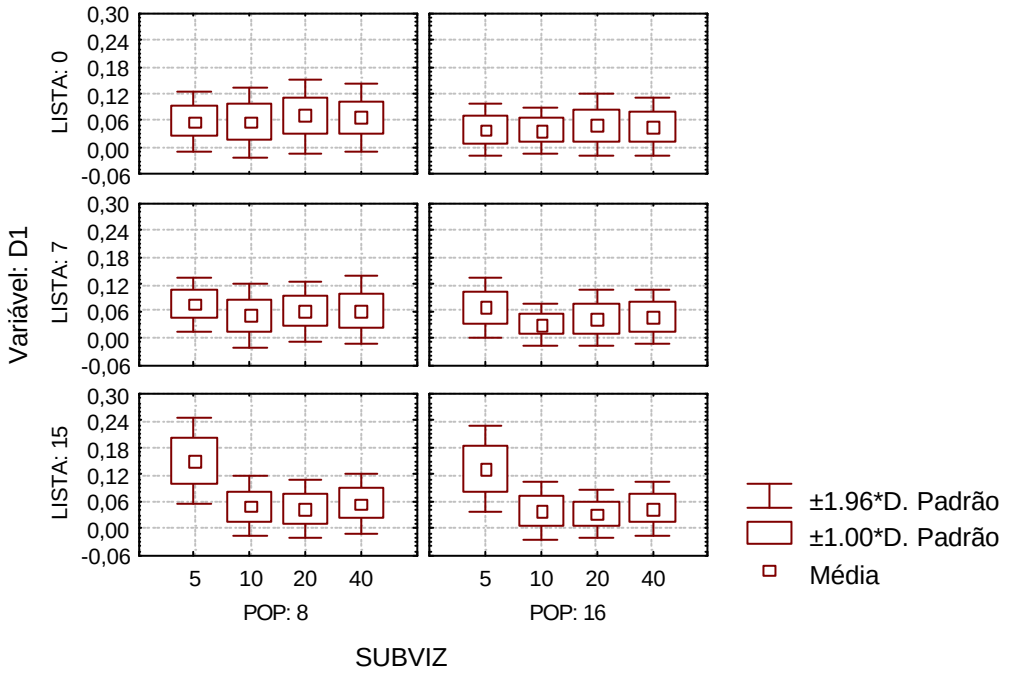


Figura 7.4 MOTS* - Diagramas do tipo caixa de D1 para *População * Lista * Subvizinhança*

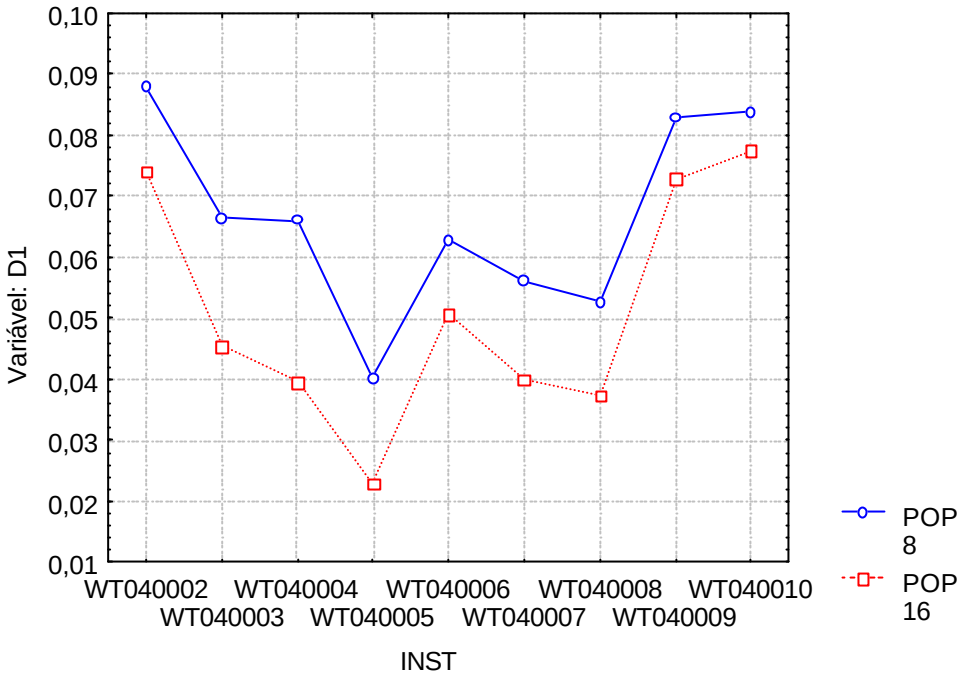


Figura 7.5 MOTS* - Diagrama de médias de D1 para *Instância * População*

Valores médios e dispersão da qualidade de aproximação

Na Tabela 7.13 são apresentados os valores médios e os desvios padrão obtidos por cada configuração, sobre o conjunto das instâncias. Acrescenta-se a esta informação o número de instâncias em que cada configuração se encontra no grupo de configurações responsáveis pelas melhores aproximações, sem que existam entre elas diferenças significativas, ao nível global (Global) e ao nível das configurações com a mesma dimensão de população (Pop.). **Apenas 7 das 24 configurações cobrem menos do que 7 instâncias, tendo 5 dessas configurações uma população com 8 soluções. Das 10 configurações que cobrem a totalidade das instâncias, 8 possuem uma população de 16 soluções.**

Um teste de correlação ordinal de *Spearman* permitiu rejeitar a hipótese de não haver uma associação entre os valores médios das distâncias e os respectivos desvios padrão, com um valor de prova de 0.001%, o que é indicativo de que **os algoritmos com melhores médias apresentam também menores dispersões**, para o conjunto das *Instâncias* (Figura 7.6).

Populações de 8 soluções						Populações de 16 soluções					
Lista tabu	**Subvizi-nhança**	**Média de D1**	**Desvio padrão de D1**	**Nº de instâncias em que é signif. melhor**		**Lista tabu**	**Subvizi-nhança**	**Média de D1**	**Desvio padrão de D1**	**Nº de instâncias em que é signif. melhor**	
				Global	**Pop.**					**Global**	**Pop.**
0	5	0.058	0.034	7	8	0	5	0.040	0.030	**9**	**9**
0	10	0.056	0.039	7	7	0	10	0.037	0.027	**9**	**9**
0	20	0.070	0.042	3	6	0	20	0.052	0.036	**9**	**9**
0	40	0.067	0.038	4	6	0	40	0.046	0.033	**9**	**9**
7	5	0.076	0.031	2	5	7	5	0.068	0.035	6	6
7	10	0.050	0.036	8	8	7	10	0.030	0.023	**9**	**9**
7	20	0.060	0.034	7	7	7	20	0.044	0.032	**9**	**9**
7	40	0.062	0.039	6	8	7	40	0.048	0.031	**9**	**9**
15	5	0.151	0.049	1	1	15	5	0.133	0.048	1	1
15	10	0.050	0.034	8	8	15	10	0.040	0.032	8	8
15	20	0.044	0.034	**9**	**9**	15	20	0.033	0.027	**9**	**9**
15	40	0.055	0.033	**9**	**9**	15	40	0.044	0.031	**9**	**9**

Tabela 7.13 MOTS* - Desempenho global ao nível da qualidade de aproximação

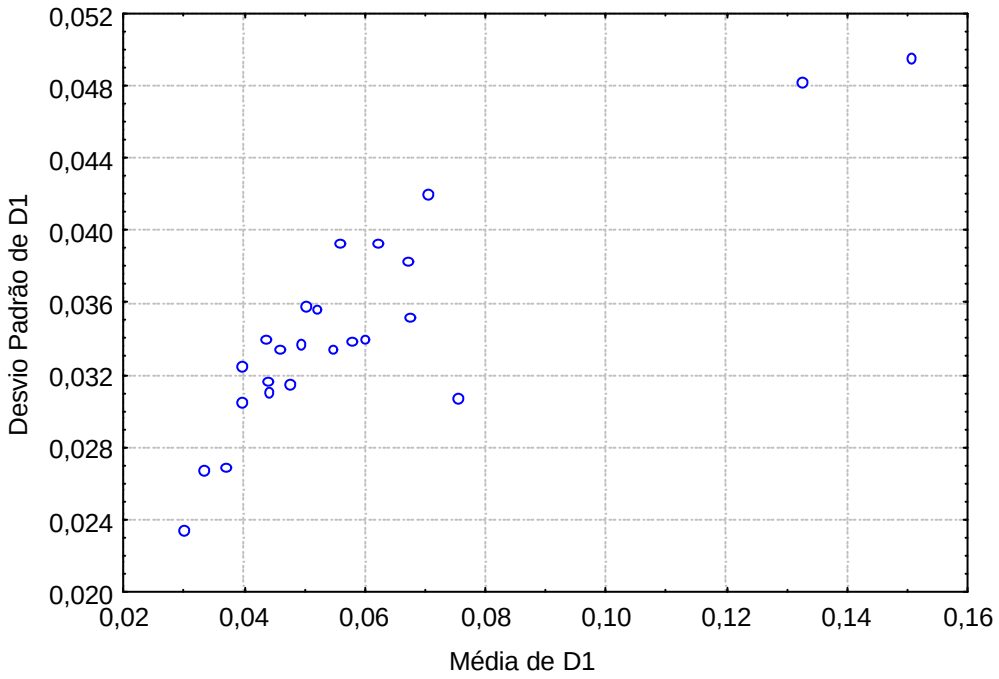


Figura 7.6 MOTS* - Diagrama de médias e desvios padrão de D1 por algoritmo

7.2.2 Tempo de execução

A tabela ANOVA para os valores do tempo de execução obtidos com as várias configurações da versão básica de MOTS* é apresentada na Tabela 7.14.

Fonte	GL Fonte	DQM Fonte	GL Erro	DQM Erro	F	Valor de prova (%)
Instância	8	2.67E+07	1944	1.08E+05	246.29	**0.00**
População	1	5.84E+07	8	3.38E+06	17.25	**0.32**
Lista	2	8.16E+06	16	4.58E+05	17.82	**0.01**
Subvizinhança	3	1.28E+07	24	9.63E+05	13.26	**0.00**
Instância * População	8	3.38E+06	1944	1.08E+05	31.22	**0.00**
Instância * Lista	16	4.58E+05	1944	1.08E+05	4.22	**0.00**
População * Lista	2	3.30E+05	16	2.09E+05	1.58	23.62
Instância * Subvizinhança	24	9.63E+05	1944	1.08E+05	8.88	**0.00**
População * Subvizinhança	3	7.43E+05	24	2.68E+05	2.77	6.33
Lista * Subvizinhança	6	1.07E+06	48	4.31E+05	2.48	**3.62**
Instância * População * Lista	16	2.09E+05	1944	1.08E+05	1.93	**1.48**
Instância * População * Subvizinhança	24	2.68E+05	1944	1.08E+05	2.47	**0.01**
Instância * Lista * Subvizinhança	48	4.31E+05	1944	1.08E+05	3.98	**0.00**
População * Lista * Subvizinhança	6	1.82E+05	48	2.76E+05	0.66	68.28
Instância * População * Lista * Subvizinhança	48	2.76E+05	1944	1.08E+05	2.54	**0.00**

Tabela 7.14 MOTS* - Tabela ANOVA para o tempo de execução

Factor Instância

Verifica-se, para o tempo de execução, uma interacção de nível 4 significativa, indicativa, em particular, de que **a interacção *População* * *Lista* * *Subvizinhança* não se mantém a mesma para as várias *Instâncias***. Nos diagramas de valores médios correspondentes (Figura 7.7), observa-se **a importância do componente de variação de proximidade dos valores médios**, já referido a respeito da mesma interacção, para D1.

As restantes interacções com o factor *Instância* são significativas, assumindo novamente importância este componente de variação do grau de proximidade.

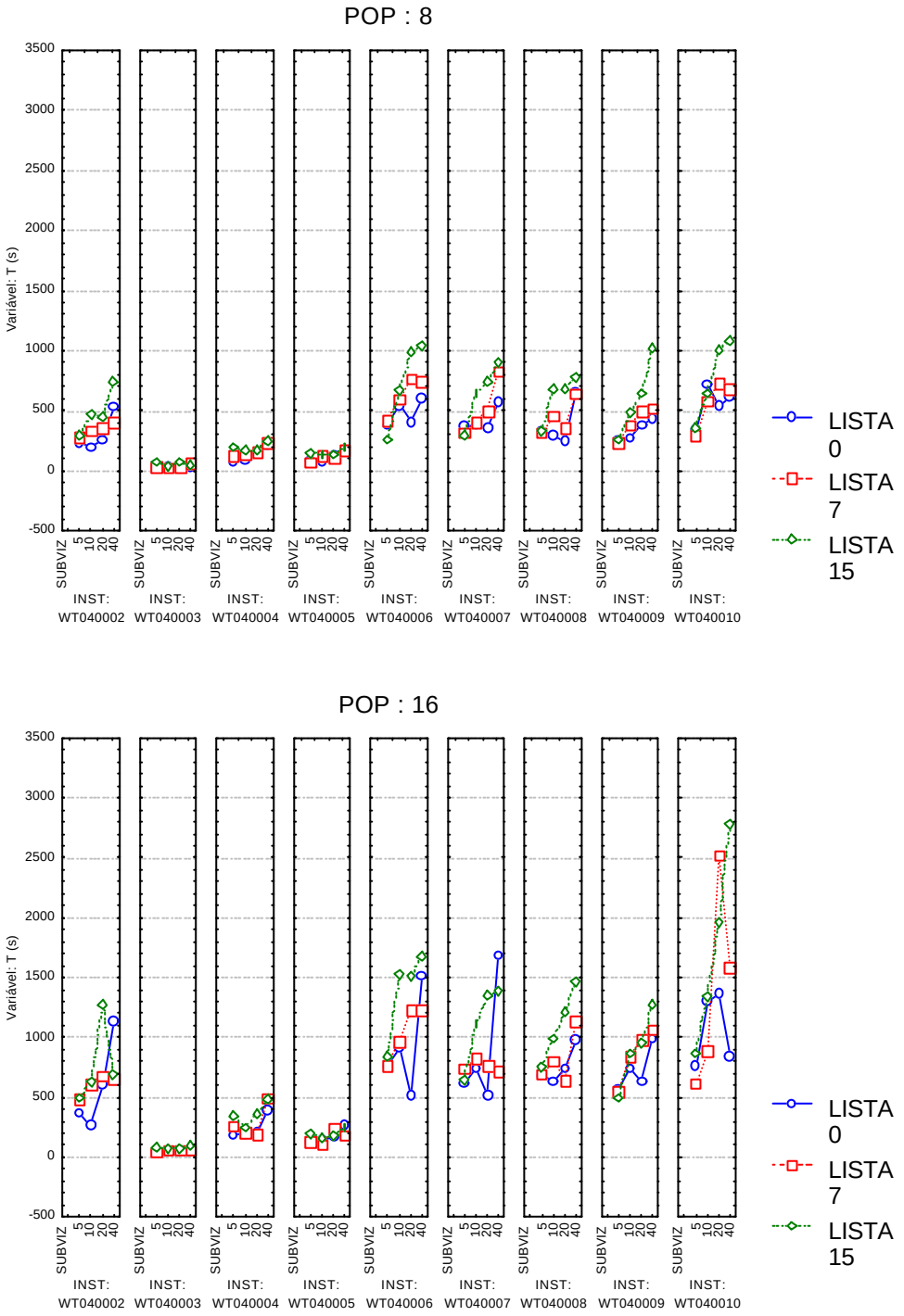


Figura 7.7 MOTS* - Diagramas de médias do tempo de execução para *Instância* * *População* * *Lista* * *Subvizinhança*

Factor População

A interacção *População * Lista * Subvizinhança* não se revela significativa para os tempos de execução, indicando a não alteração da interacção *Lista * Subvizinhança* com a variação da dimensão da *População* (Figura 7.8). **O aumento da dimensão da *População* conduz a um aumento dos tempos de execução médios, mas sem que haja alteração significativa da interacção *Lista * Subvizinhança*.** É, ainda, notório (Figura 7.9) um **acréscimo da dispersão dos resultados obtidos com *Populações* de 16 soluções, bem como com o aumento da dimensão da *Subvizinhança***.

Para a interacção, também significativa, entre *Instância* e *População*, a interpretação do diagrama de médias da Figura 7.10 permite verificar que, **em todas as *Instâncias*, se obtêm, para *Populações* de 16 soluções, tempos de execução médios superiores**.

As diferenças entre os tempos de execução médios para as duas dimensões de *População* variam significativamente consoante a *Instância*, ocorrendo as maiores diferenças para maiores valores de tempos de execução médios. O teste de Tukey para as diferenças de valor esperado confirma a existência de diferenças significativas entre os tempos de execução médios relativos às dimensões de população consideradas, para todas as *Instâncias*, à excepção das 3, 4 e 5.

A interacção *População * Lista* não é significativa, sendo observável (Figura 7.11) um aumento sensivelmente constante dos tempos de execução com o aumento da dimensão da *População*. Já a interacção *População * Subvizinhança* é praticamente significativa, verificando-se que o acréscimo de tempo de execução se agrava significativamente para, simultaneamente, maiores dimensões de *População* e *Subvizinhança* (Figura 7.12).

Interacção Lista * Subvizinhança

A interacção *Lista * Subvizinhança* é significativa, existindo, no entanto, um claro significado individual para os efeitos, conforme se pode observar na Figura 7.9. De uma forma geral, **o tempo de execução aumenta quer com a dimensão da *Lista*, quer com a dimensão da *Subvizinhança***.

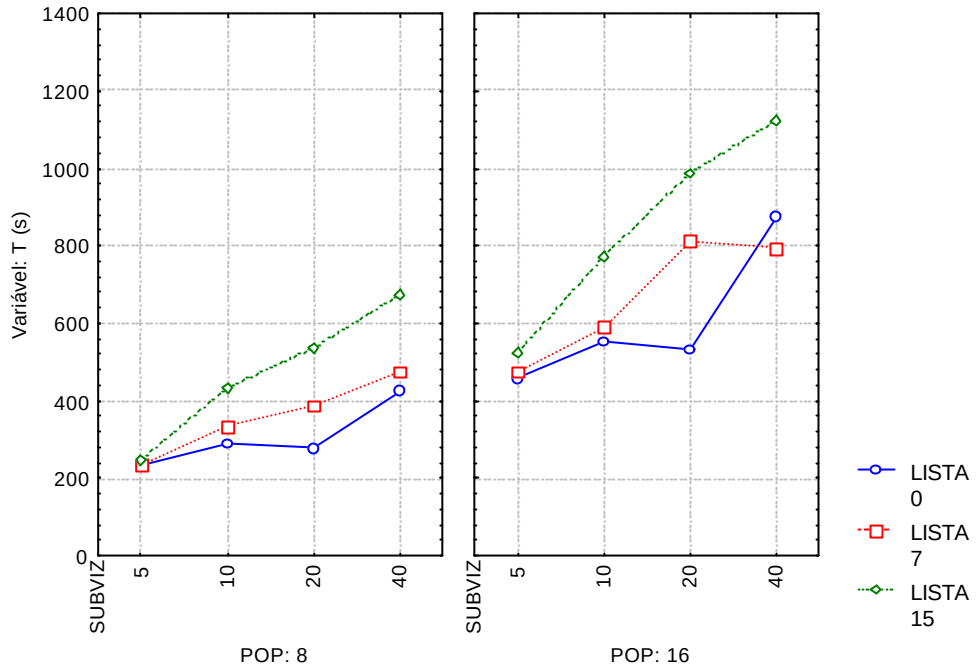


Figura 7.8 MOTS* - Diagrama de médias do tempo de execução para *População* * *Lista* * *Subvizinhança*

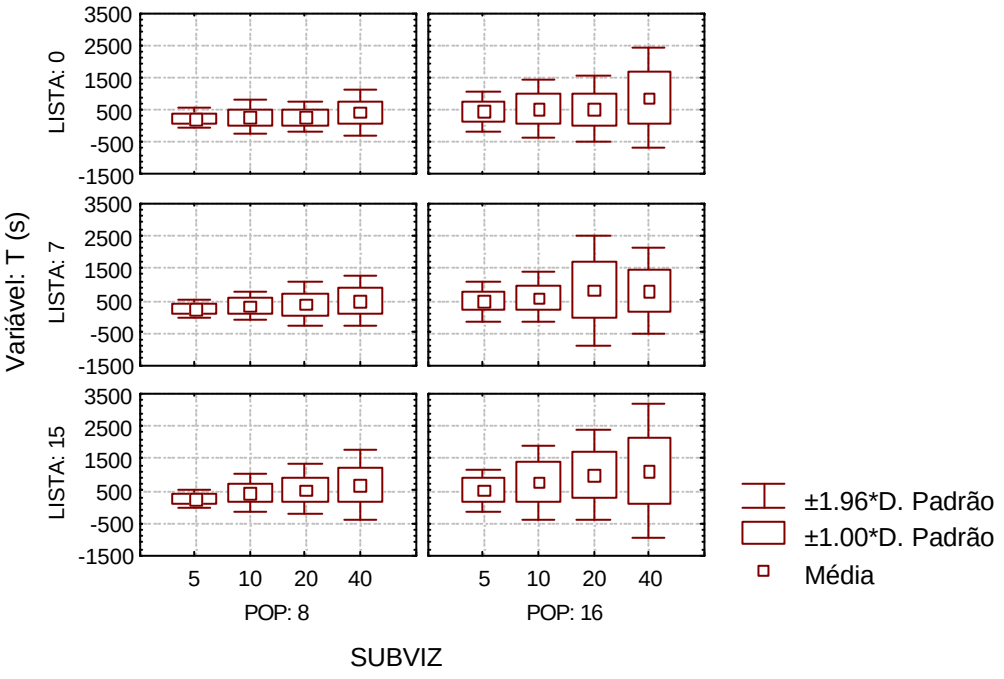


Figura 7.9 MOTS* - Diagrama do tipo caixa do tempo de execução para *População* * *Lista* * *Subvizinhança*

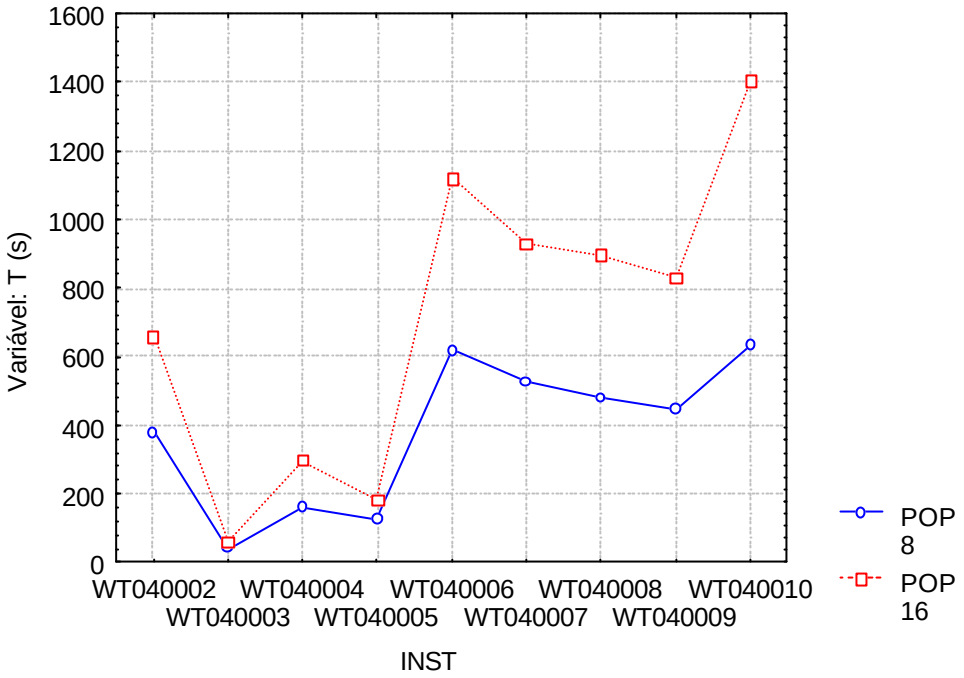


Figura 7.10 MOTS* - Diagrama de médias do tempo de execução para *Instância * População*

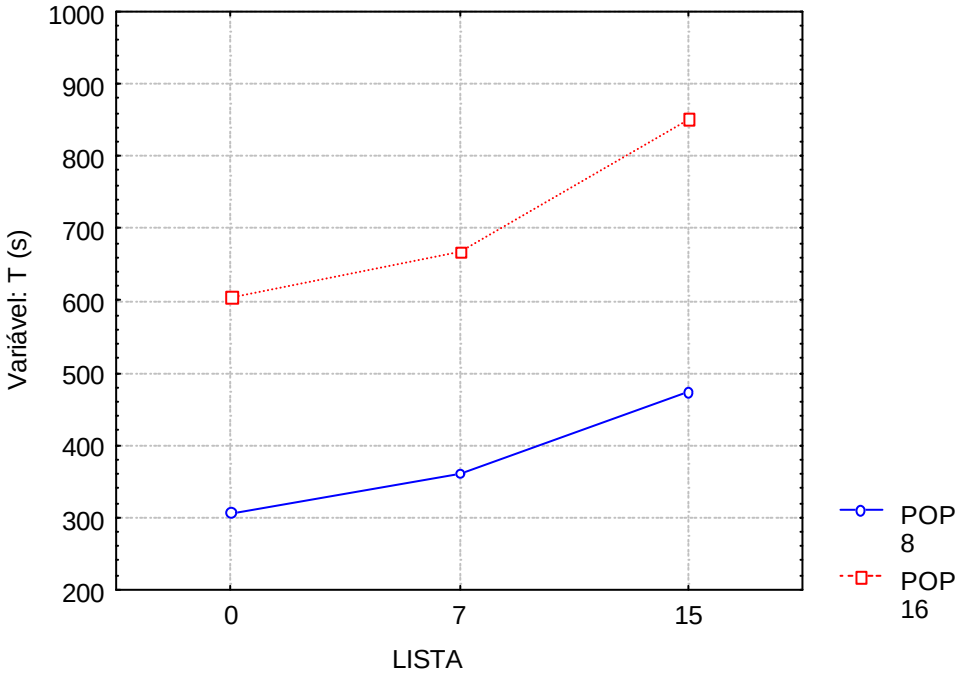


Figura 7.11 MOTS* - Diagrama de médias do tempo de execução para *População * Lista*

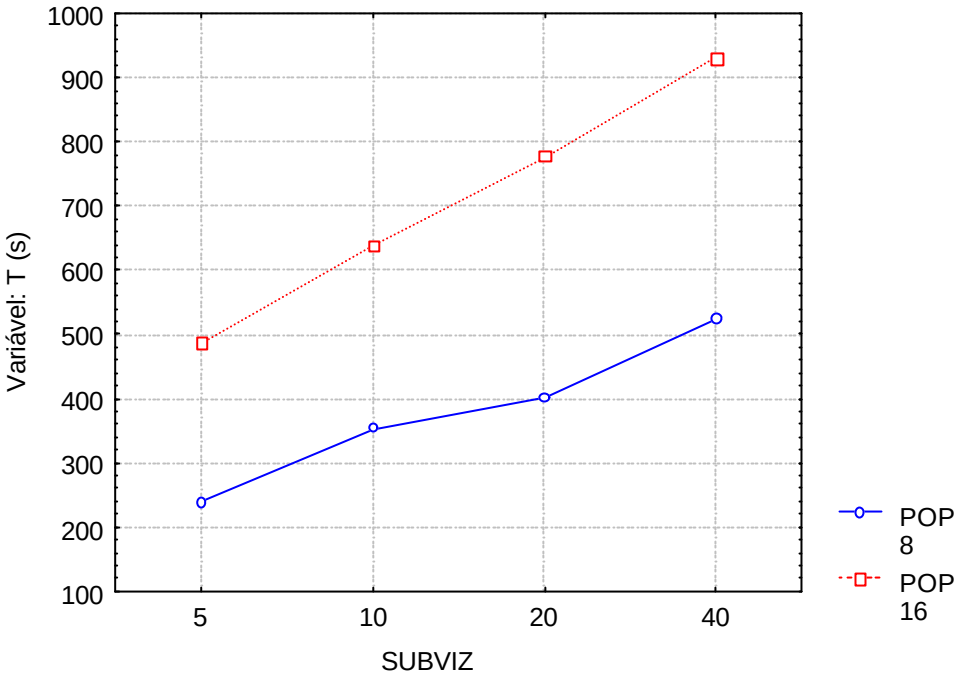


Figura 7.12 MOTS* - Diagrama de médias do tempo de execução para *População* * *Subvizinhança*

Valores médios e dispersão dos tempos de execução

A Tabela 7.15 apresenta os valores médios e os desvios padrão obtidos por cada configuração, sobre o conjunto das instâncias, bem como o número de instâncias em que cada configuração se encontra no grupo de configurações responsáveis pelos menores tempos de execução, sem que existam entre elas diferenças significativas, ao nível global (Global) e ao nível das configurações com a mesma dimensão de população (Pop.). **Apenas 9 das 24 configurações cobrem menos do que 7 instâncias, tendo 6 dessas configurações uma população com 16 soluções. Das 13 configurações que cobrem a totalidade das instâncias, 10 possuem uma população de 8 soluções. As restantes 3 são configurações com a menor dimensão considerada para a subvizinhança.**

O teste de correlação ordinal de *Spearman* permitiu testar a associação entre os valores médios dos tempos de execução e os respectivos desvios padrão. A hipótese de não existir tal associação foi rejeitada, com um valor de prova de 0.00%, o que é indicativo de que **os algoritmos com maiores valores médios apresentam também maiores dispersões**, para o conjunto das *Instâncias* (Figura 7.13).

Este teste foi igualmente utilizado para verificar a associação entre os valores médios dos tempos de execução e o número de soluções não-dominadas de cada *Instância*. A hipótese de inexistência de tal associação foi rejeitada, com um valor de prova de 0.00%, indicando que

as *Instâncias* com maior número de soluções não-dominadas exigem maiores tempos de execução (Figura 7.14).

Populações de 8 soluções						Populações de 16 soluções					
Lista tabu	Subvizi-nhança	Média de T (s)	Desvio padrão de T (s)	Nº de instâncias em que é signif. menor		Lista tabu	Subvizi-nhança	Média de T (s)	Desvio padrão de T (s)	Nº de instâncias em que é signif. menor	
				Global	Pop.					Global	Pop.
0	5	233.44	158.36	**9**	**9**	0	5	458.18	319.39	**9**	**9**
0	10	288.58	254.97	**9**	**9**	0	10	554.70	456.74	7	8
0	20	277.73	241.17	**9**	**9**	0	20	533.04	512.41	7	8
0	40	424.28	355.23	**9**	**9**	0	40	874.21	793.58	4	6
7	5	234.72	152.15	**9**	**9**	7	5	475.99	311.15	**9**	**9**
7	10	336.62	227.30	**9**	**9**	7	10	591.91	397.02	7	**9**
7	20	389.38	350.41	**9**	**9**	7	20	812.66	868.29	5	7
7	40	476.10	388.98	**9**	**9**	7	40	793.87	669.35	4	7
15	5	247.07	137.52	**9**	**9**	15	5	523.36	333.53	**9**	**9**
15	10	434.71	290.44	**9**	**9**	15	10	772.69	584.59	4	7
15	20	538.65	398.39	6	7	15	20	985.15	717.26	3	5
15	40	672.03	548.30	5	6	15	40	1122.55	1044.15	3	4

Tabela 7.15 MOTS* - Desempenho global ao nível do tempo de execução

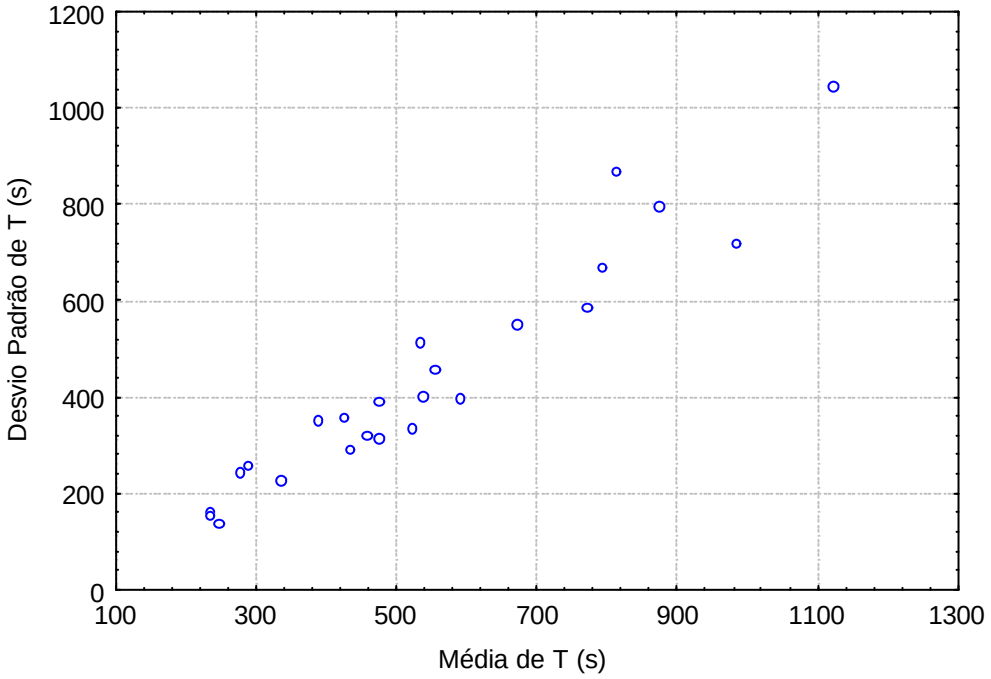


Figura 7.13 MOTS* - Diagrama de médias e desvios padrão do tempo de execução por algoritmo

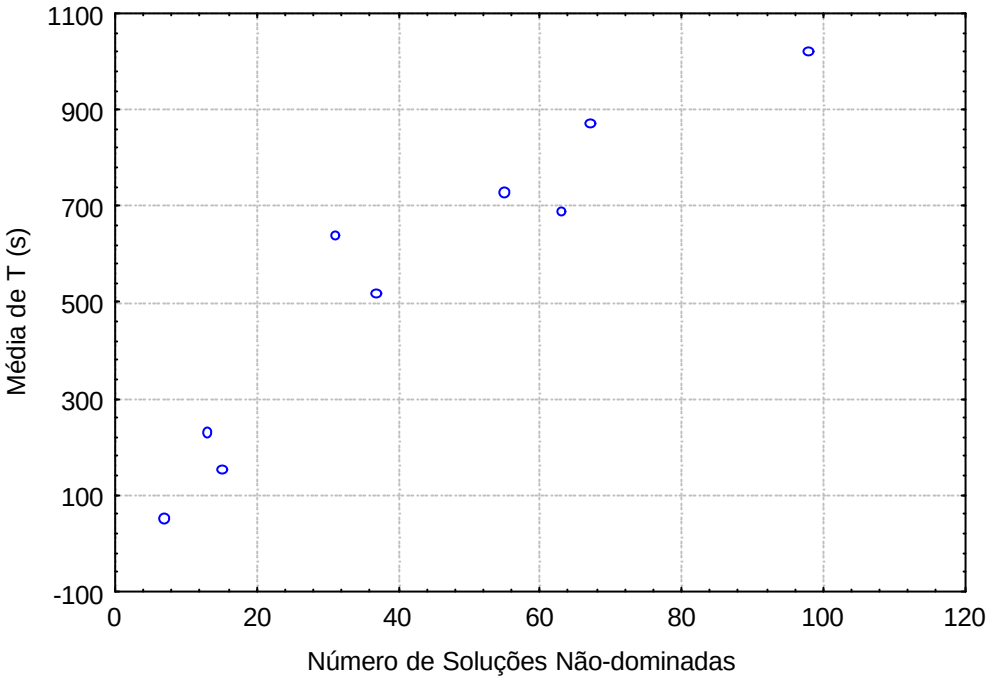


Figura 7.14 MOTS* - Diagrama de tempo de execução médio e número de soluções não-dominadas por instância

7.2.3 Análise conjunta

Na Figura 7.15 apresenta-se um diagrama com os valores médios dos tempos de execução e das distâncias D1 para cada algoritmo considerado.

Verifica-se, de uma forma geral, um **agrupamento dos algoritmos com *Populações* de dimensão 8 numa área de tempos de execução mais reduzidos e distâncias maiores. Para os algoritmos com *Populações* de dimensão 16, o agrupamento realiza-se numa região simétrica, de tempos de execução superiores e distâncias D1 menores**. O teste de correlação ordinal de Spearman, à associação entre qualidade e tempo de execução permitiu rejeitar a hipótese de inexistência de tal associação, com um valor de prova de 0.02%. **É, portanto, estatisticamente significativo o aumento da qualidade (redução da distância D1) com o aumento do tempo de execução**.

Os algoritmos eficientes, em termos de resultados médios, são os seguintes: (8, 0, 5), (8, 0, 10), (8, 7, 10), (8, 15, 10), (16, 0, 5), (16, 0, 10) e (16, 7, 10). Apenas o algoritmo (16, 0, 5) cobre a totalidade das instâncias no grupo dos algoritmos com, simultaneamente, melhor qualidade de aproximação e menor tempo de execução.

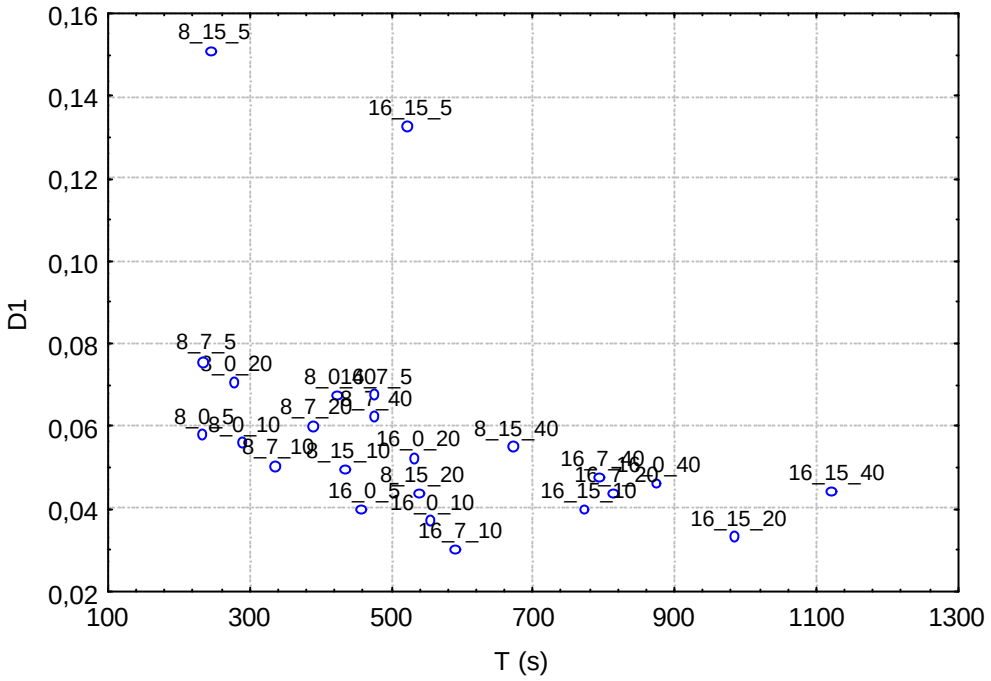


Figura 7.15 MOTS* - Diagrama de qualidade de aproximação e tempo de execução por algoritmo

7.3 Versão básica de PSA

7.3.1 Qualidade de aproximação

A tabela ANOVA para os valores de D1 obtidos com as várias configurações da versão básica de PSA é apresentada na Tabela 7.16.

Fonte	GL Fonte	DQM Fonte	GL Erro	DQM Erro	F	Valor de prova (%)
Instância	8	0.2081	1944	0.0030	69.4647	**0.00**
População	1	0.4512	8	0.0128	35.1783	**0.03**
Temperatura	3	0.1825	24	0.0116	15.6635	**0.00**
Plateau	2	0.2556	16	0.0163	15.6845	**0.02**
Instância * População	8	0.0128	1944	0.0030	4.2814	**0.00**
Instância * Temperatura	24	0.0116	1944	0.0030	3.8888	**0.00**
População * Temperatura	3	0.0042	24	0.0034	1.2515	31.32
Instância * Plateau	16	0.0163	1944	0.0030	5.4390	**0.00**
População * Plateau	2	0.0036	16	0.0031	1.1765	33.36
Temperatura * Plateau	6	0.0771	48	0.0096	8.0157	**0.00**
Instância * População * Temperatura	24	0.0034	1944	0.0030	1.1185	31.33
Instância * População * Plateau	16	0.0031	1944	0.0030	1.0195	43.20
Instância * Temperatura * Plateau	48	0.0096	1944	0.0030	3.2107	**0.00**
População * Temperatura * Plateau	6	0.0038	48	0.0022	1.7233	13.59
Instância * População * Temperatura * Plateau	48	0.0022	1944	0.0030	0.7454	90.17

Tabela 7.16 PSA - Tabela ANOVA para D1

Factor Instância

Graficamente seria observável, para valores médios, um efeito da *Instância* sobre a interacção *População* * *Temperatura* * *Plateau* semelhante ao que ocorre com MOTS*. No entanto, esta interacção não é estatisticamente significativa, verificando-se, com recurso a testes de Tukey, que, **em cada *Instância*, praticamente não existem diferenças significativas entre as configurações algorítmicas consideradas**.

Essas diferenças surgem significativas ao nível da interacção *Instância* * *Temperatura* * *Plateau*, a única de nível 3 que se apresenta significativa, denotando, em específico, que **a interacção *Temperatura* * *Plateau* se modifica entre as várias *Instâncias***. Com a realização de testes de Tukey, verifica-se que, **à excepção das configurações com *Temperatura* 2 e *Plateau* 50 e 200, e *Temperatura* 0.5 e *Plateau* 200, não existe evidência estatística de diferenças entre as várias configurações consideradas**, para as várias *Instâncias*. Para as configurações enumeradas, é observável (Figura 7.16) o componente de variação da proximidade referido nas análises do factor *Instância* com a versão básica de MOTS*.

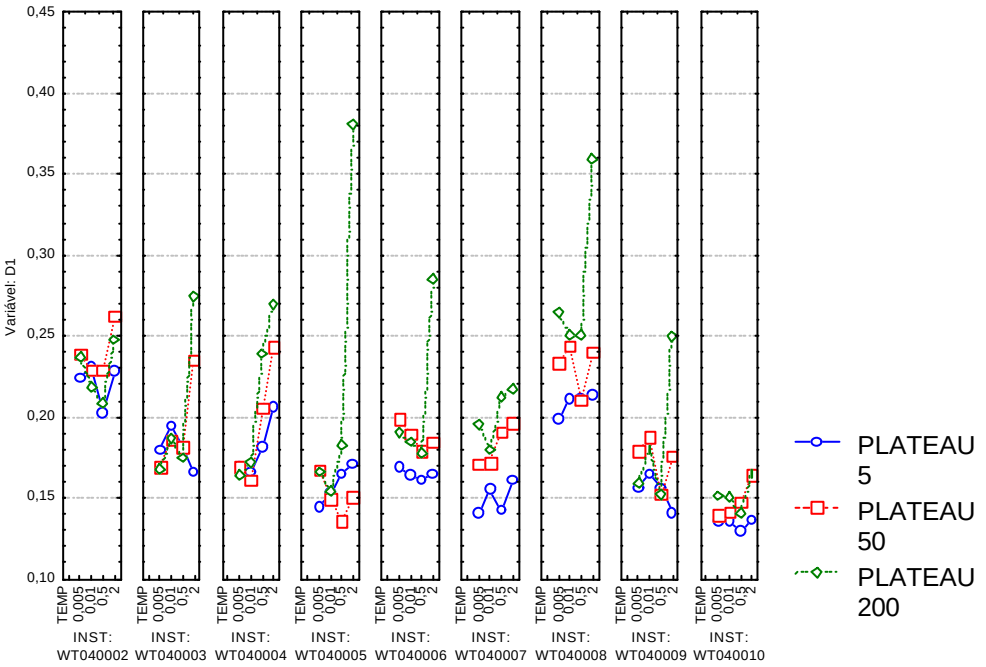


Figura 7.16 PSA - Diagrama de médias de D1 para *Instância* * *Temperatura* * *Plateau*

Factor População

A menos de flutuações aleatórias, **o aumento da dimensão da *População* conduz a uma melhoria das aproximações, mas sem alteração significativa da interacção *Temperatura* * *Plateau***, conforme indicado pelo carácter não significativo da interacção que

envolve os três factores. O diagrama de médias da Figura 7.17 **não sugere a existência de aproximação dos desempenhos médios, induzida pelo aumento da população**.

No diagrama do tipo caixa da Figura 7.18 é observável um **ligeiro decréscimo da dispersão dos resultados obtidos com as configurações englobando *Populações* de 16 soluções**.

No diagrama de médias da Figura 7.19 é possível verificar que, **em todas as *Instâncias*, se obtêm melhores resultados com *Populações* de 16 soluções**. Da interacção significativa é observável que a diferença entre os resultados médios para as duas dimensões de *População* varia conforme a *Instância*. Uma análise *post hoc*, baseada nos testes de *Tukey* para a igualdade de valor esperado, permitiu verificar que existem diferenças significativas entre os resultados obtidos para as dimensões de *População* consideradas para todas as *Instâncias*, à excepção das 3, 4 e 10.

Interacção Temperatura * Plateau

A interacção *Temperatura* * *Plateau* é significativa, verificando-se com clareza (Figura 7.17 e Figura 7.18) apenas que, por um lado, **um arrefecimento rápido conduz a melhores aproximações**, e por outro, **temperaturas iniciais muito elevadas conduzem a piores aproximações**. **Nos restantes níveis, as diferenças não são significativas**, de acordo com os testes de Tukey referidos a propósito da interacção *Instância* * *Temperatura* * *Plateau*.

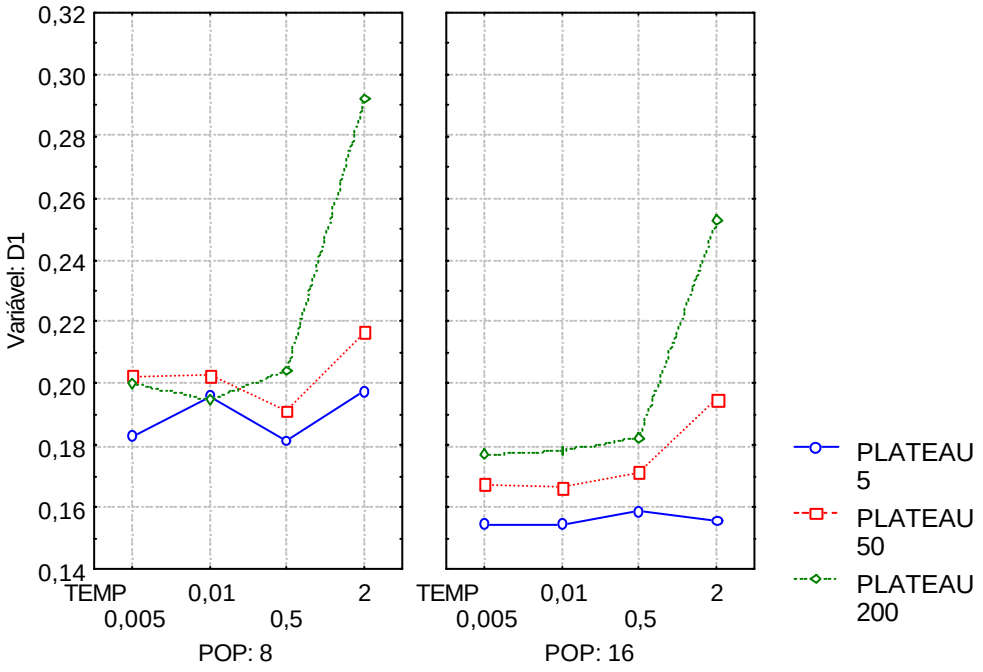


Figura 7.17 PSA - Diagrama de médias de D1 para *População* * *Temperatura* * *Plateau*

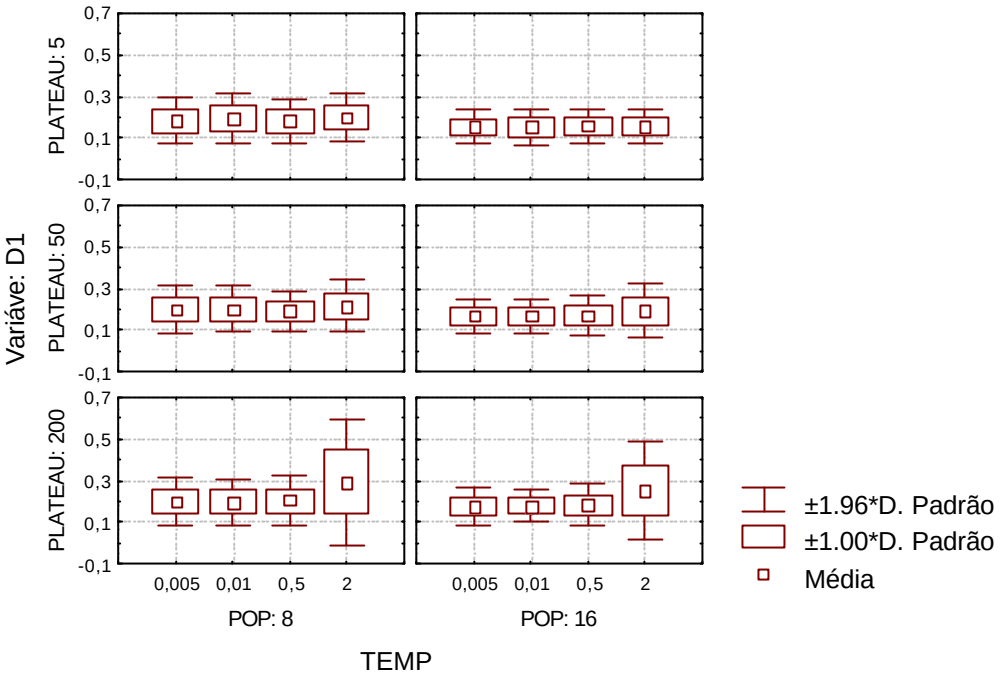


Figura 7.18 PSA - Diagrama do tipo caixa de D1 para *População* * *Temperatura* * *Plateau*

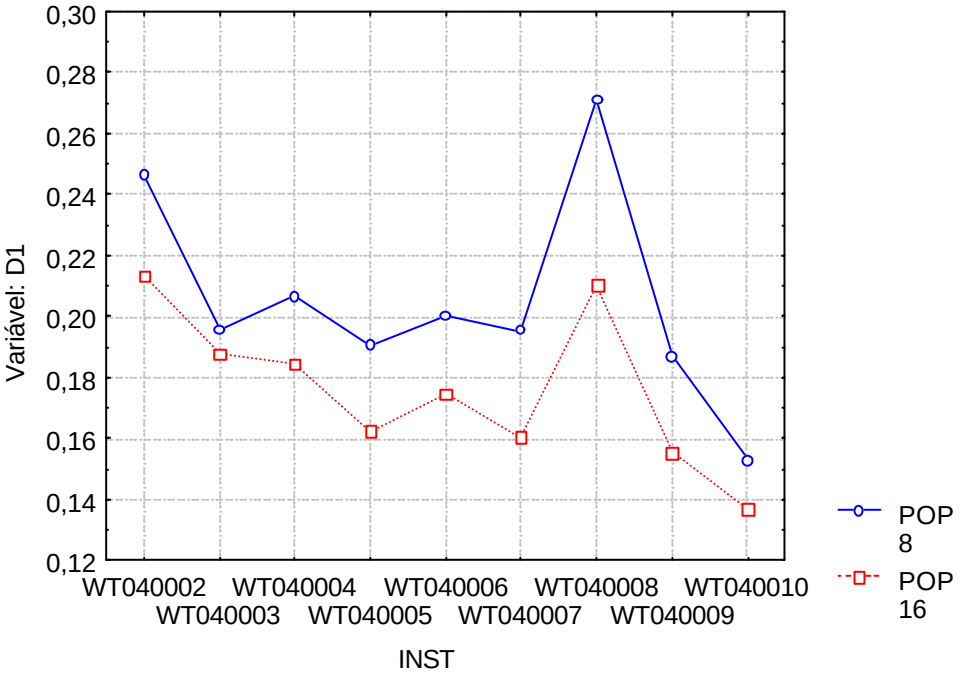


Figura 7.19 PSA - Diagrama de médias de D1 para *Instância* * *População*

Valores médios e dispersão da qualidade de aproximação

Na Tabela 7.17 são apresentados os valores médios e os desvios padrão obtidos por cada configuração, sobre o conjunto das instâncias, conjuntamente com o número de instâncias em que cada configuração se encontra no grupo de configurações responsáveis pelas melhores aproximações, sem que existam entre elas diferenças significativas, ao nível global (Global) e ao nível das configurações com a mesma dimensão de população (Pop.). **Apenas são excluídas deste grupo as configurações com *Temperatura* 2 e *Plateau* 50 e 200, bem como as configurações com 8 soluções e *Plateau* 200. Apenas duas configurações apresentam uma cobertura de menos de 8 instâncias**.

Um teste de correlação ordinal de *Spearman* permitiu rejeitar a hipótese de inexistência de associação entre os valores médios das distâncias e os respectivos desvios padrão, com um valor de prova de 0.00%, o que é indicativo de que **os algoritmos com melhores médias apresentam também menores dispersões**, para o conjunto das *Instâncias* (Figura 7.20).

Populações de 8 soluções						**Populações de 16 soluções**					
Temp. inicial	***Plateau***	**Média de D1**	**Desvio padrão de D1**	**N° de instâncias em que é signif. melhor**		**Temp. inicial**	***Plateau***	**Média de D1**	**Desvio padrão de D1**	**N° de instâncias em que é signif. melhor**	
				Global	**Pop.**					**Global**	**Pop.**
0.005	5	0.183	0.057	**9**	**9**	0.005	5	0.154	0.041	**9**	**9**
0.005	50	0.202	0.059	**9**	**9**	0.005	50	0.168	0.041	**9**	**9**
0.005	200	0.200	0.058	8	**9**	0.005	200	0.177	0.046	**9**	**9**
0.01	5	0.196	0.061	**9**	**9**	0.01	5	0.154	0.043	**9**	**9**
0.01	50	0.203	0.057	**9**	**9**	0.01	50	0.166	0.041	**9**	**9**
0.01	200	0.195	0.056	8	**9**	0.01	200	0.178	0.038	**9**	**9**
0.5	5	0.181	0.055	**9**	**9**	0.5	5	0.159	0.041	**9**	**9**
0.5	50	0.191	0.051	**9**	**9**	0.5	50	0.171	0.049	**9**	**9**
0.5	200	0.204	0.060	8	**9**	0.5	200	0.183	0.052	**9**	**9**
2.0	5	0.198	0.059	**9**	**9**	2.0	5	0.155	0.042	**9**	**9**
2.0	50	0.216	0.063	8	8	2.0	50	0.195	0.068	8	8
2.0	200	0.292	0.155	2	4	2.0	200	0.253	0.121	5	5

Tabela 7.17 PSA - Desempenho global ao nível da qualidade de aproximação

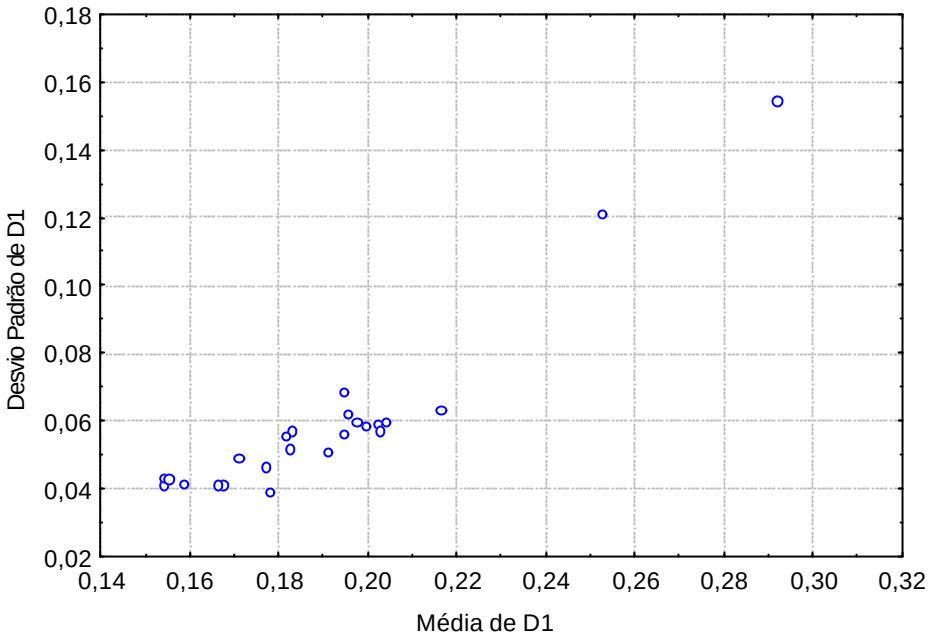


Figura 7.20 PSA - Diagrama de médias e desvios padrão de D1 por algoritmo

7.3.2 Tempo de execução

A tabela ANOVA para os valores do tempo de execução obtidos com as várias configurações da versão básica de PSA é apresentada na Tabela 7.18.

Fonte	GL Fonte	DQM Fonte	GL Erro	DQM Erro	F	Valor de prova (%)
Instância	8	5.26E+05	1944	2.44E+03	215.55	**0.00**
População	1	2.97E+06	8	7.55E+04	39.37	**0.02**
Temperatura	3	4.07E+05	24	1.27E+04	32.05	**0.00**
Plateau	2	3.11E+05	16	1.99E+04	15.62	**0.02**
Instância * População	8	7.55E+04	1944	2.44E+03	30.95	**0.00**
Instância * Temperatura	24	1.27E+04	1944	2.44E+03	5.20	**0.00**
População * Temperatura	3	5.59E+04	24	4.61E+03	12.12	**0.00**
Instância * Plateau	16	1.99E+04	1944	2.44E+03	8.17	**0.00**
População * Plateau	2	2.88E+04	16	5.16E+03	5.59	**1.45**
Temperatura * Plateau	6	1.21E+05	48	8.98E+03	13.48	**0.00**
Instância * População * Temperatura	24	4.61E+03	1944	2.44E+03	1.89	**0.57**
Instância * População * Plateau	16	5.16E+03	1944	2.44E+03	2.12	**0.60**
Instância * Temperatura * Plateau	48	8.98E+03	1944	2.44E+03	3.68	**0.00**
População * Temperatura * Plateau	6	1.25E+04	48	3.70E+03	3.37	**0.75**
Instância * População * Temperatura * Plateau	48	3.70E+03	1944	2.44E+03	1.52	**1.31**

Tabela 7.18 PSA - Tabela ANOVA para o tempo de execução

Factor Instância

Ao inverso do sucedido com a distância D1, para o tempo de execução verifica-se a existência de uma interacção significativa de nível 4, indicativa, em particular, de que **a interacção *População* * *Temperatura* * *Plateau* não se mantém a mesma para as várias *Instâncias***. Com efeito, todas as interacções que envolvem o factor *Instância* são significativas, verificando-se em todas a **importância do componente de variação do grau de proximidade dos valores médios**, referido nas anteriores análises das interacções que envolvem este factor.

Factor População

Novamente de modo inverso ao verificado com D1, a interacção *População* * *Temperatura* * *Plateau* revela-se significativa para os tempos de execução, com um **maior afastamento relativo dos resultados nas configurações com 16 soluções** (diagrama de médias da Figura 7.21). Além de um aumento dos valores médios dos tempos de execução, é, ainda, notório (diagrama do tipo caixa da Figura 7.22) um **acréscimo da dispersão dos resultados obtidos com *Populações* de 16 soluções, bem como com o aumento da dimensão do *Plateau***.

Para a interacção, também significativa, entre *Instância* e *População*, o diagrama de médias da Figura 7.23 permite verificar que, **em todas as *Instâncias*, se obtêm, para *Populações* de 16 soluções, tempos de execução médios superiores**. **As diferenças entre os tempos de execução médios para as duas dimensões de *População* variam significativamente consoante a *Instância*, verificando-se que as maiores diferenças ocorrem para maiores valores dos tempos de execução médios**.

O teste de Tukey para as diferenças de valor esperado confirma a existência de diferenças significativas entre os tempos de execução médios relativos às dimensões de população consideradas, para todas as *Instâncias*, excepto a 3.

As interacções dos factores individuais com *População* são significativas, verificando-se que **o acréscimo de tempo de execução se agrava significativamente para, simultaneamente, valores crescentes da *População* e dos factores individuais** (Figura 7.24 e Figura 7.25).

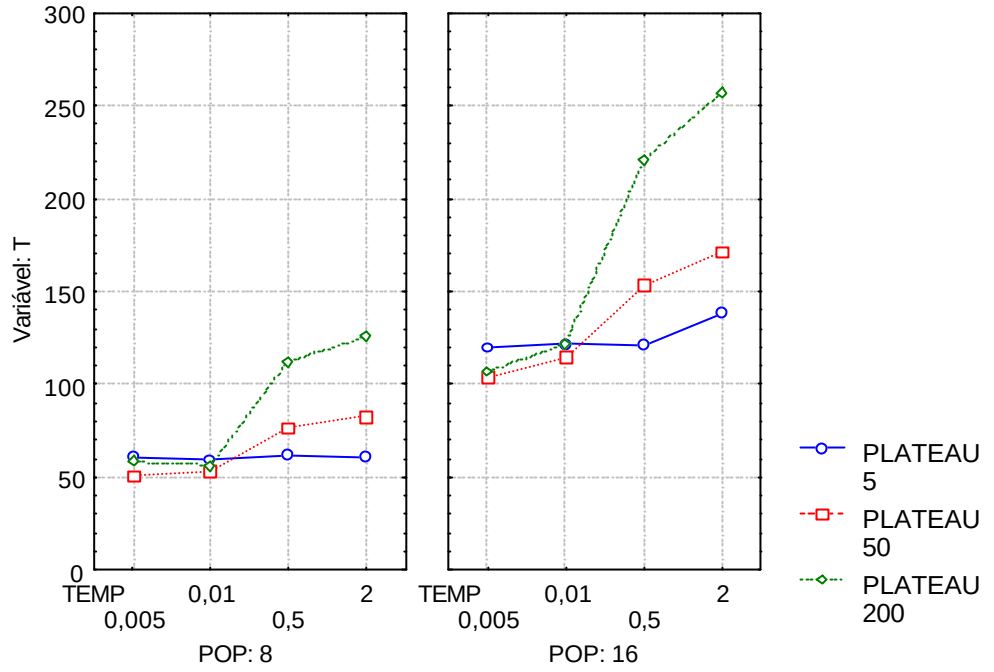


Figura 7.21 PSA - Diagrama de médias do tempo de execução para *População*

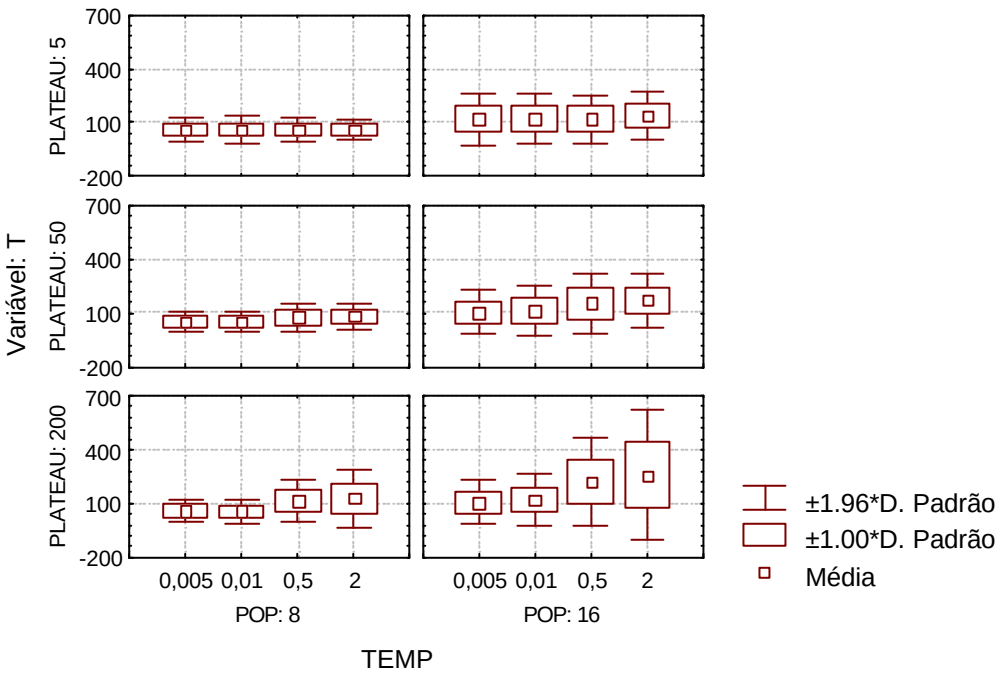


Figura 7.22 PSA - Diagrama do tipo caixa do tempo de execução para *População *Temperatura * Plateau*

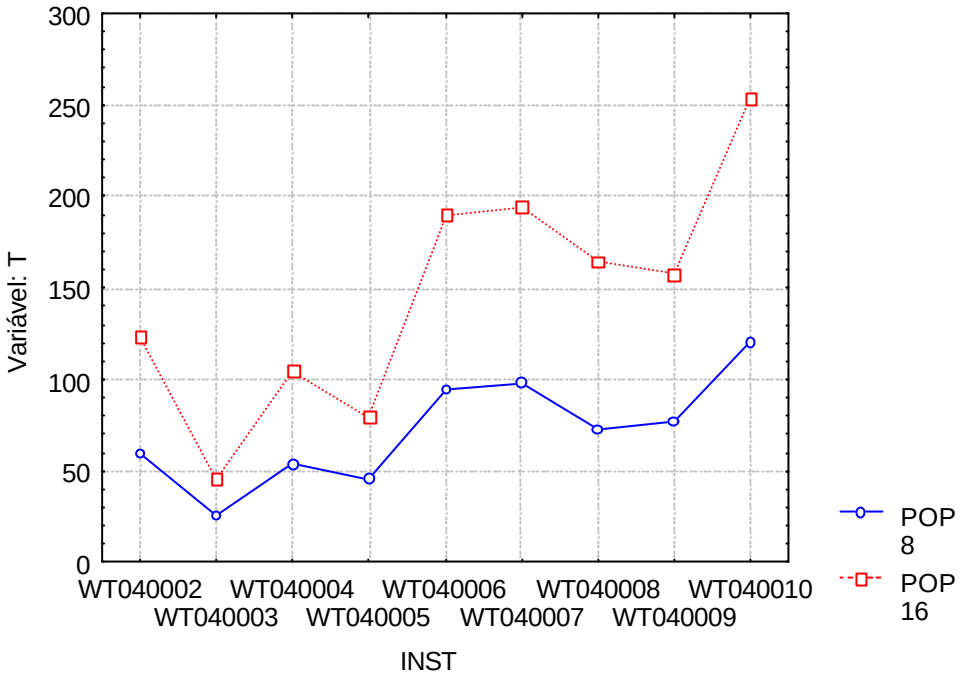


Figura 7.23 PSA - Diagrama de médias do tempo de execução para *Instância* * *População*

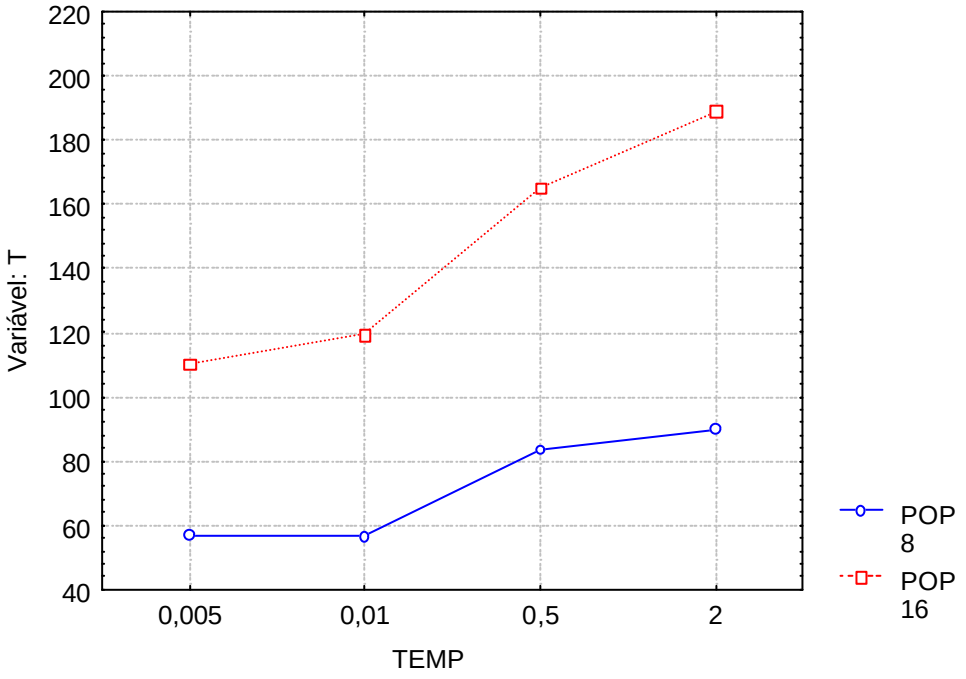


Figura 7.24 PSA - Diagrama de médias do tempo de execução para *População* * *Temperatura*

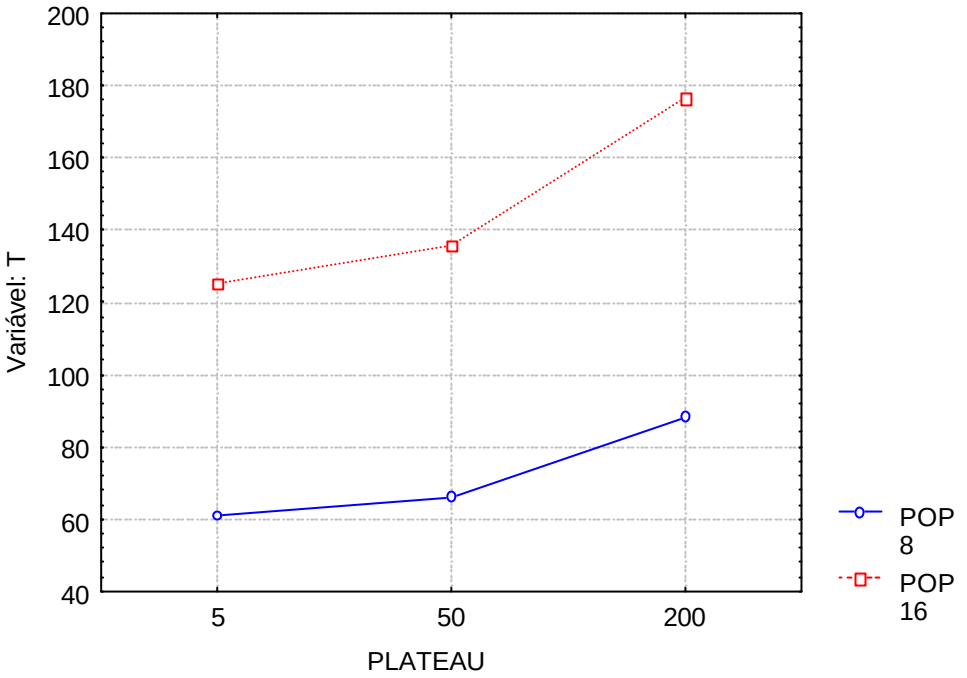


Figura 7.25 PSA - Diagrama de médias do tempo de execução para *População * Plateau*

Interacção Temperatura * Plateau

A interacção *Temperatura * Plateau* é significativa, existindo, no entanto, um claro significado individual para os efeitos (Figura 7.21 e Figura 7.22). **De uma forma geral, o tempo de execução aumenta quer com a *Temperatura*, quer com a dimensão do *Plateau*. Para *Plateaus* muito reduzidos, a variação com a *Temperatura* é pouco significativa**.

Valores médios e dispersão dos tempos de execução

A Tabela 7.19 apresenta os valores médios e os desvios padrão obtidos por cada configuração, sobre o conjunto das instâncias, bem como o número de instâncias em que cada configuração se encontra no grupo de configurações responsáveis pelos menores tempos de execução, sem que existam entre elas diferenças significativas, ao nível global (Global) e ao nível das configurações com a mesma dimensão de população (Pop.). **Este conjunto de algoritmos apenas inclui configurações com 8 soluções, excluindo as configurações com *Plateau* 200 e *Temperaturas* 0.5 e 2.0. Das 24 configurações, 10 apresentam coberturas de menos de 8 instâncias, sendo que destas apenas 2 possuem populações de 8 soluções**.

O teste de correlação ordinal de *Spearman* permitiu testar a associação entre os valores médios dos tempos de execução e as respectivas dispersões. A hipótese de não haver uma

associação entre os valores médios dos tempos de execução e a respectiva dispersão foi rejeitada, com um valor de prova de 0.00%, o que é indicativo de que **os algoritmos com maiores valores médios apresentam também maiores dispersões**, para o conjunto das *Instâncias* (Figura 7.26).

O teste de correlação ordinal de *Spearman* permitiu igualmente testar a associação entre os valores médios dos tempos de execução e o número de soluções não-dominadas de cada *Instância*. A hipótese de inexistência de associação entre os tempos de execução médios e o número de soluções não-dominadas foi rejeitada, com um valor de prova de 0.05%, indicando que **as *Instâncias* com maior número de soluções não-dominadas exigem maiores tempos de execução** (Figura 7.27).

Populações de 8 soluções						**Populações de 16 soluções**					
Temp. inicial	***Plateau***	**Média de T (s)**	**Desvio padrão de T (s)**	**Nº de instâncias em que é signif. menor**		**Temp. inicial**	***Plateau***	**Média de T (s)**	**Desvio padrão de T(s)**	**Nº de instâncias em que é signif. menor**	
				Global	**Pop.**					**Global**	**Pop.**
0.005	5	61.09	35.48	**9**	**9**	0.005	5	119.51	72.87	6	**9**
0.005	50	51.23	29.78	**9**	**9**	0.005	50	103.93	61.69	8	**9**
0.005	200	58.65	31.80	**9**	**9**	0.005	200	106.84	62.72	8	**9**
0.01	5	59.71	37.82	**9**	**9**	0.01	5	121.71	69.40	6	**9**
0.01	50	53.54	27.55	**9**	**9**	0.01	50	114.92	71.56	8	**9**
0.01	200	55.93	33.19	**9**	**9**	0.01	200	121.49	73.78	6	**9**
0.5	5	61.81	36.39	**9**	**9**	0.5	5	121.28	68.61	8	**9**
0.5	50	76.55	41.35	**9**	**9**	0.5	50	153.57	85.48	4	7
0.5	200	112.19	58.56	7	7	0.5	200	220.91	127.19	2	5
2.0	5	61.02	31.10	**9**	**9**	2.0	5	138.39	66.50	7	8
2.0	50	83.02	37.98	**9**	**9**	2.0	50	171.22	76.34	2	8
2.0	200	125.84	84.61	5	5	2.0	200	257.24	184.38	2	5

Tabela 7.19 PSA - Desempenho global ao nível do tempo de execução

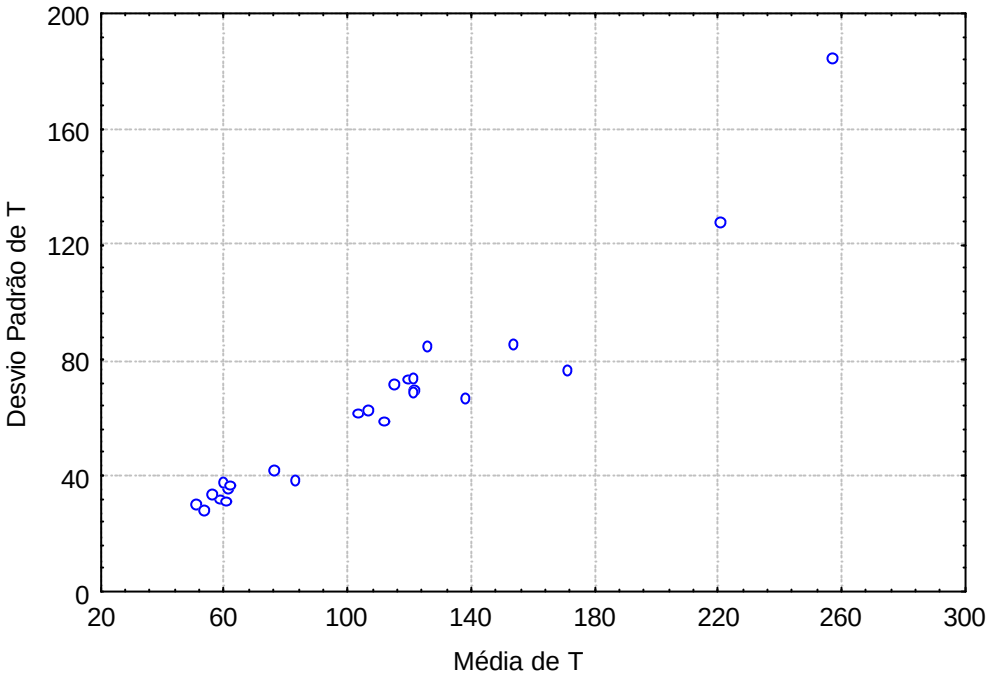


Figura 7.26 PSA - Diagrama de médias e desvios padrão do tempo de execução por algoritmo

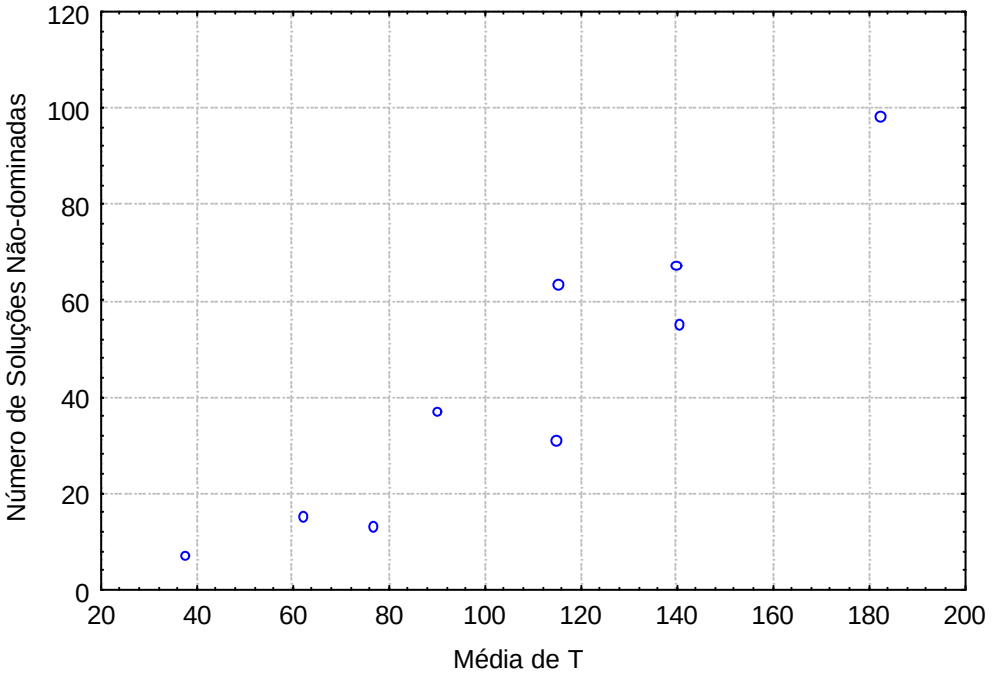


Figura 7.27 PSA - Diagrama de tempo de execução médio e número de soluções não-dominadas por instância

7.3.3 Análise conjunta

O diagrama com os valores médios dos tempos de execução e das distâncias D1, para cada algoritmo considerado, é apresentado na Figura 7.28. Tal como para a MOTS*,

verifica-se, de uma forma geral, um **agrupamento dos algoritmos com *Populações* de dimensão 8 numa área de tempos de execução mais reduzidos e distâncias maiores, enquanto que, para os algoritmos com *Populações* de dimensão 16, o agrupamento se realiza numa região simétrica, de tempos de execução superiores e distâncias D1 menores**.

Não foi possível rejeitar a hipótese de inexistência de associação entre qualidade e tempo de execução, no âmbito de um teste de correlação ordinal de Spearman, com um valor de prova de 18.37%.

Os algoritmos eficientes, em termos de resultados médios, são os seguintes: (8, 0.005, 5), (8, 0.005, 50), (8, 0.01, 200), (8, 0.5, 5), (16, 0.005, 5), (16, 0.005, 50) e (16, 0.01, 50). Os algoritmos que cobrem a totalidade das instâncias no grupo dos algoritmos com, simultaneamente, melhor qualidade de aproximação e menor tempo de execução, são os seguintes: (8, 0.005, 5), (8, 0.005, 50), (8, 0.01, 5), (8, 0.01, 50), (8, 0.5, 5), (8, 0.5, 50) e (8, 2.0, 5).

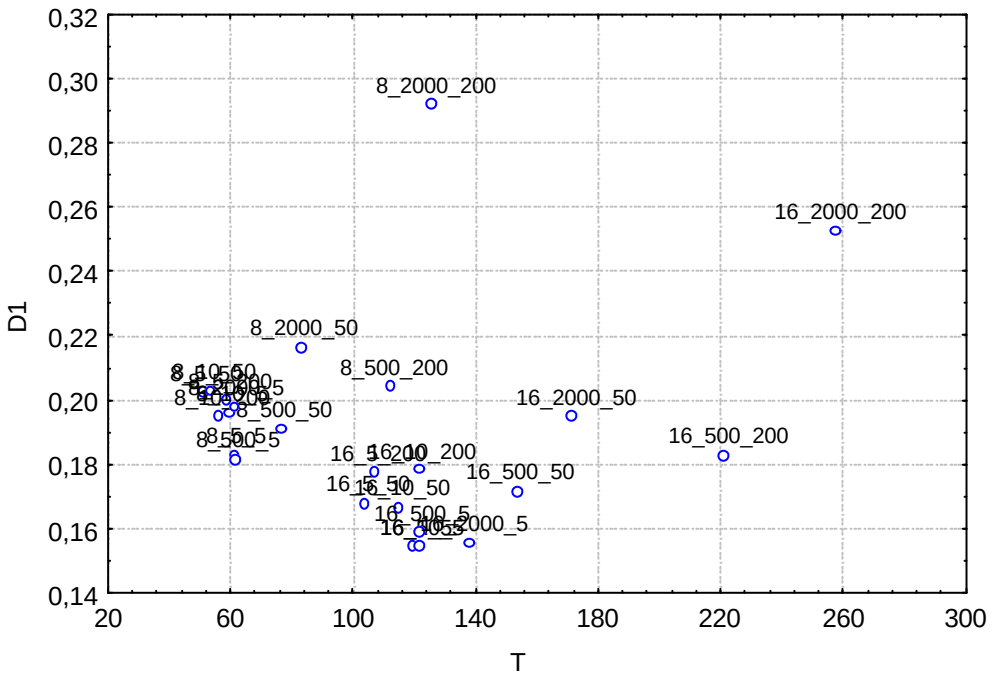


Figura 7.28 PSA - Diagrama de qualidade de aproximação e tempo de execução por algoritmo

7.4 PSA com subvizinhanças

Conforme referido em 7.1.5 (Configuração dos algoritmos), este conjunto de testes será conduzido sobre as configurações básicas de PSA que apresentam o melhor e o pior desempenhos médios, ao nível da qualidade de aproximação, ou seja, respectivamente (A, 1), com (16, 0.005, 5), e (B, 1), com (8, 2, 200).

As tabelas ANOVA para os valores de D1 e do tempo de execução obtidos com as várias configurações de PSA com subvizinhanças são apresentadas, respectivamente, na Tabela 7.20 e na Tabela 7.21.

Fonte	GL Fonte	DQM Fonte	GL Erro	DQM Erro	F	Valor de prova (%)
Instância	8	0.0550	648	0.0033	16.5550	**0.00**
Configuração	1	0.4371	8	0.0145	30.1012	**0.06**
Subvizinhança	3	0.7895	24	0.0136	58.0853	**0.00**
Instância * Configuração	8	0.0145	648	0.0033	4.3717	**0.00**
Instância * Subvizinhança	24	0.0136	648	0.0033	4.0918	**0.00**
Configuração * Subvizinhança	3	0.1718	24	0.0081	21.2496	**0.00**
Instância * Configuração * Subvizinhança	24	0.0081	648	0.0033	2.4343	**0.02**

Tabela 7.20 PSA com subvizinhanças - Tabela ANOVA para D1

Fonte	GL Fonte	DQM Fonte	GL Erro	DQM Erro	F	Valor de prova (%)
Instância	8	1.41E+06	648	1.12E+04	125.2047	**0.00**
Configuração	1	5.49E+05	8	6.43E+04	8.5402	**1.92**
Subvizinhança	3	1.47E+06	24	1.69E+05	8.7025	**0.04**
Instância * Configuração	8	6.43E+04	648	1.12E+04	5.7267	**0.00**
Instância * Subvizinhança	24	1.69E+05	648	1.12E+04	15.0437	**0.00**
Configuração * Subvizinhança	3	3.89E+05	24	5.62E+04	6.9242	**0.16**
Instância * Configuração * Subvizinhança	24	5.62E+04	648	1.12E+04	5.0023	**0.00**

Tabela 7.21 PSA com subvizinhanças - Tabela ANOVA para o tempo de execução

Para ambos os aspectos em estudo, qualidade de aproximação e tempo de execução, todos os factores principais e interacções se revelam significativos.

Apenas para a configuração B é estatisticamente significativa a melhoria de qualidade de aproximação, com o aumento da dimensão da *Subvizinhança*, observável no diagrama de médias da Figura 7.29. Uma análise *post hoc*, com testes de Tukey, permite concluir que o grupo de algoritmos de maior qualidade, e sem diferenças significativas entre eles em todas as instâncias, apenas exclui (B, 1) e (B, 2). Por outro lado, graficamente (Figura 7.31), para a configuração A não se observam alterações assinaláveis de dispersão, enquanto para a configuração B tal sucede a partir da dimensão 2 para a *Subvizinhança*.

No diagrama de médias do tempo de execução (Figura 7.30), é observável o respectivo aumento com o aumento da dimensão da *Subvizinhança*, superior para a configuração A. Uma análise *post hoc*, com testes de Tukey, permite concluir que o grupo de algoritmos com menores tempos de execução, e sem diferenças significativas entre eles em todas as instâncias, é composto por todos os algoritmos com dimensões de sub-vizinhança 1 e 2. O algoritmo (B, 5) está ausente deste conjunto em apenas uma instância. Relativamente à

dispersão dos tempos de execução, observa-se no diagrama do tipo caixa da Figura 7.32 um aumento da dispersão com o aumento do tempo de execução, tal como em testes anteriores.

De uma forma geral, verifica-se que os compromissos qualidade de aproximação / tempo de execução mais interessantes se mantêm na configuração base A. Relativamente à configuração base B, o caso com subvizinhança de dimensão 5 será o único a apresentar-se vantajoso nas duas perspectivas.

Para as configurações e tipos de instâncias analisados, a consideração de subvizinhanças em PSA não conduziu à obtenção de resultados de qualidade significativamente superior à melhor configuração sem subvizinhanças, mas permitiu colocar o desempenho da pior configuração ao nível da melhor, quer na perspectiva da qualidade, quer na perspectiva do tempo de execução. A consideração deste tipo de estratégia poderá, portanto, apresentar-se vantajosa do ponto de vista da robustez em relação às configurações base do PSA, embora com uma contrapartida ao nível do incremento dos tempos de execução.

Será ainda de destacar que com *Subvizinhanças* de dimensão 5 e 10 os níveis de desempenho do PSA se começam a aproximar dos níveis da MOTS*, na área de compromisso com menor qualidade de aproximação e menor tempo de execução.

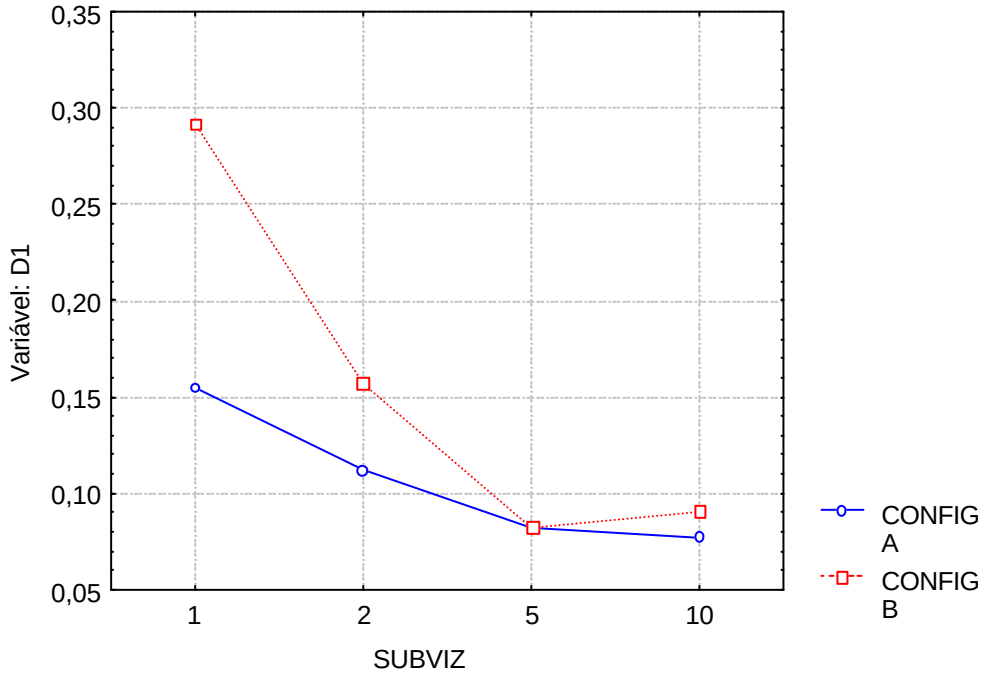


Figura 7.29 PSA com subvizinhanças - Diagrama de médias de D1 para *Configuração * Subvizinhança*

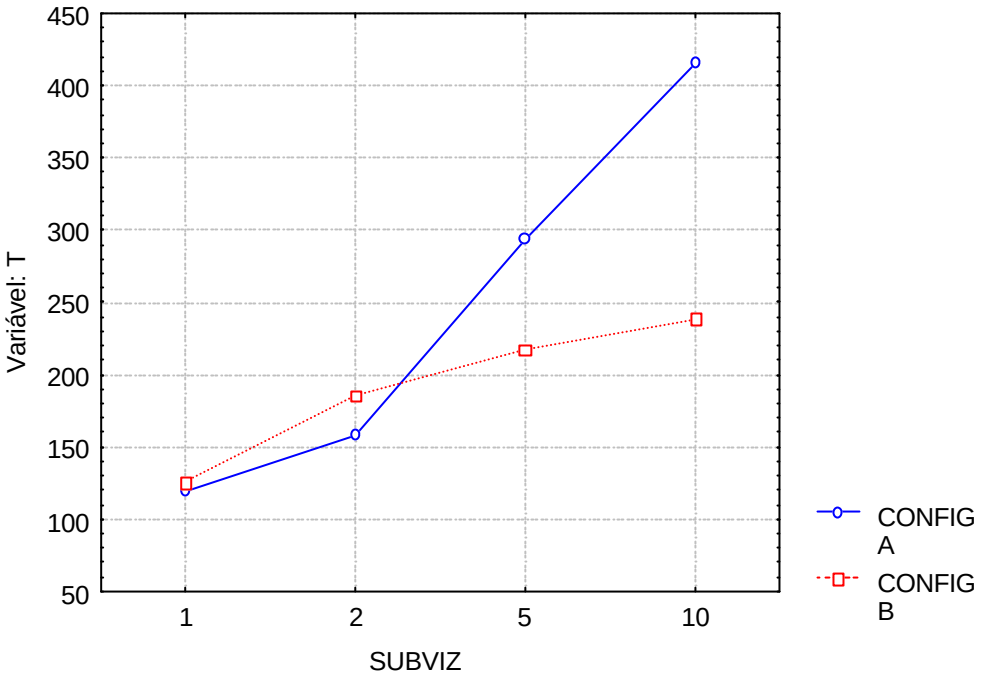


Figura 7.30 PSA com subvizinhanças - Diagrama de médias do tempo de execução para *Configuração * Subvizinhança*

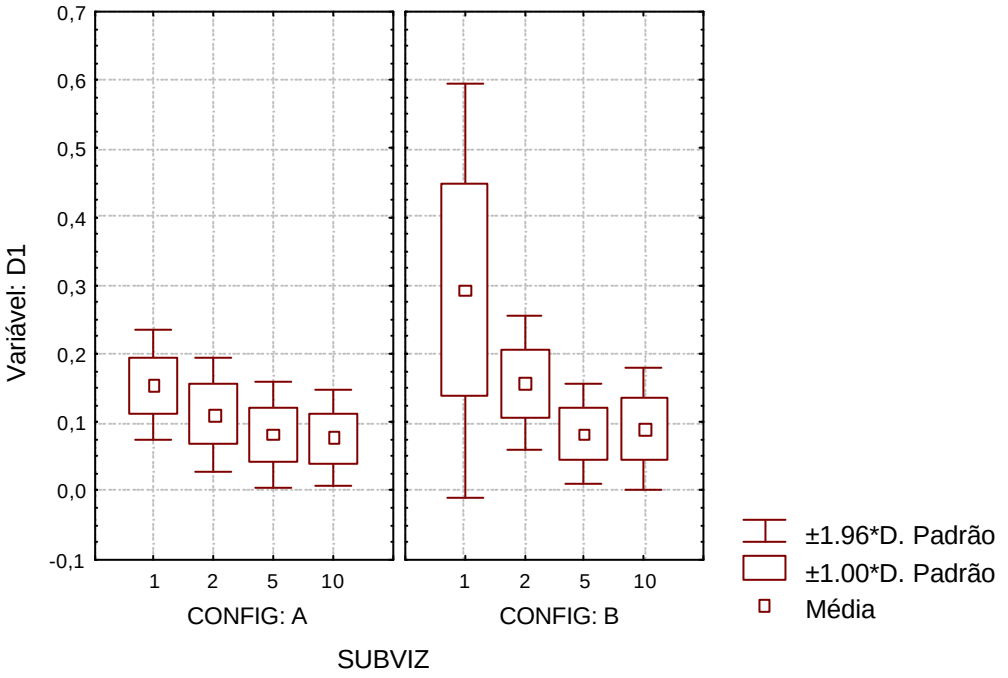


Figura 7.31 PSA com subvizinhanças - Diagrama do tipo caixa de D1 para *Configuração * Subvizinhança*

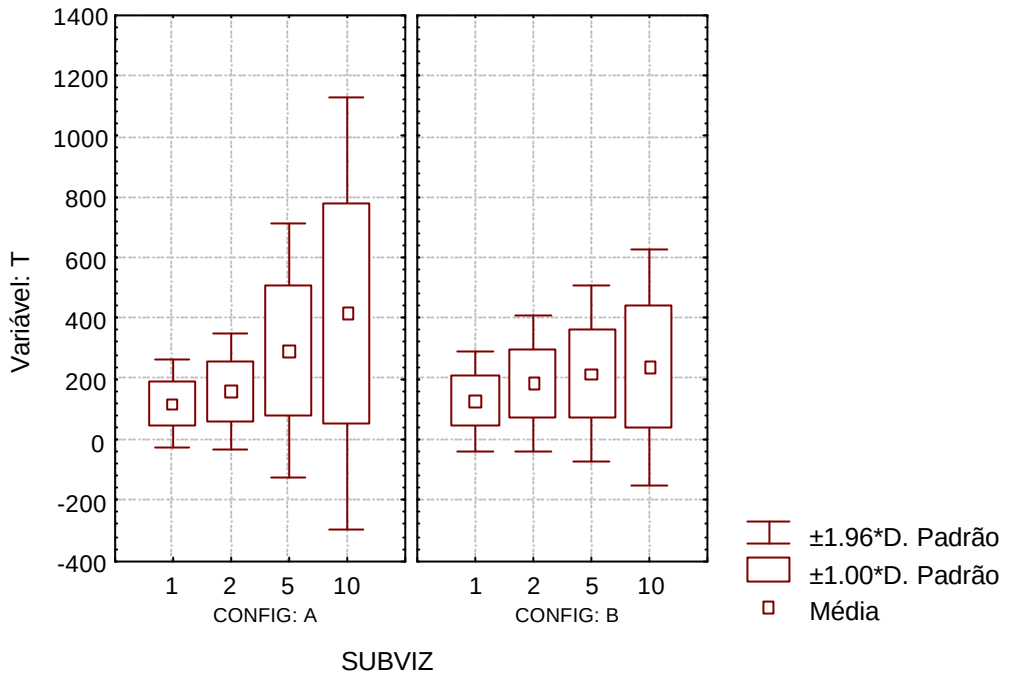


Figura 7.32 PSA com subvizinhanças - Diagrama do tipo caixa do tempo de execução para *Configuração * Subvizinhança*

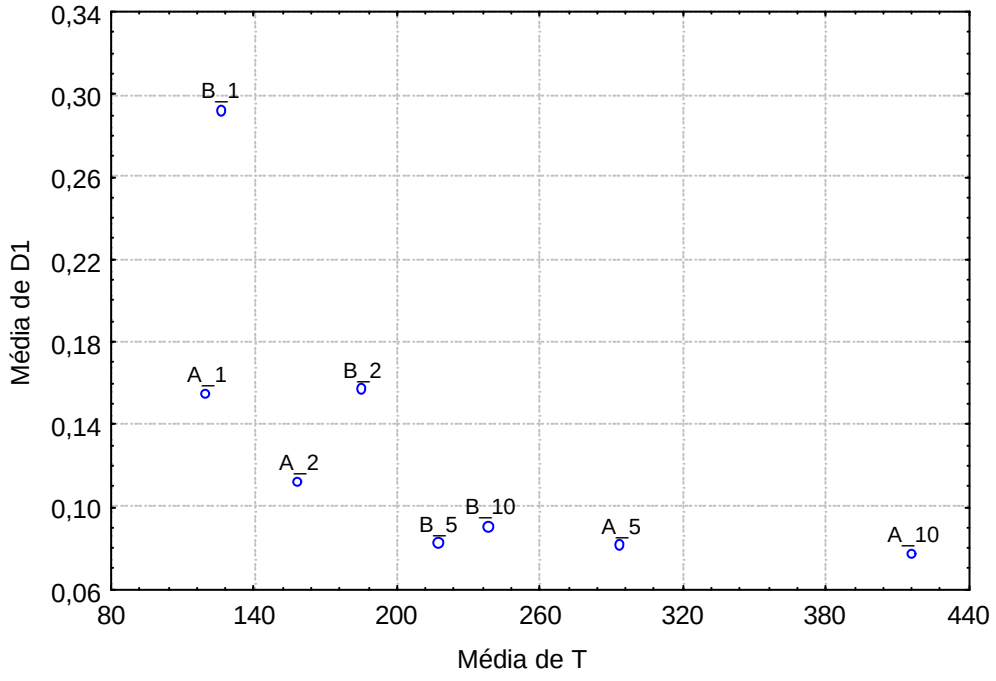


Figura 7.33 PSA com subvizinhanças - Diagrama de qualidade de aproximação e tempo de execução por algoritmo

Configuração	Subvizinhança	D1				T			
		Média	Desvio padrão	Nº de instâncias em que é signif. melhor		Média (s)	Desvio padrão (s)	Nº de instâncias em que é signif. menor	
				Global	Config.			Global	Config.
A	1	0.154	0.041	**9**	**9**	119.51	72.87	**9**	**9**
A	2	0.112	0.043	**9**	**9**	158.44	98.88	**9**	**9**
A	5	0.082	0.039	**9**	**9**	293.76	215.21	4	5
A	10	0.077	0.036	**9**	**9**	415.81	362.32	4	4
B	1	0.292	0.155	0	0	125.84	84.61	**9**	**9**
B	2	0.157	0.050	6	6	185.36	113.41	**9**	**9**
B	5	0.082	0.037	**9**	**9**	216.96	149.27	8	8
B	10	0.090	0.046	**9**	**9**	238.40	199.21	5	5

Tabela 7.22 PSA com subvizinhanças - Desempenho global

7.5 PSA com lista de candidatos

Tal como referido em 7.1.5, este conjunto de testes será conduzido sobre as configurações de PSA com subvizinhanças que apresentam o melhor e o pior desempenhos médios, ao nível da qualidade de aproximação, ou seja, respectivamente (A, 1), com (16, 0.005, 5) e subvizinhança 10, e (B, 1), com (8, 2, 200) e subvizinhança 2.

As tabelas ANOVA para os valores de D1 e do tempo de execução, obtidos com as várias configurações de PSA com lista de candidatos, são apresentadas, respectivamente, na Tabela 7.23 e na Tabela 7.24.

Fonte	GL Fonte	DQM Fonte	GL Erro	DQM Erro	F	Valor de prova (%)
Instância	8	0.0511	648	0.0008	61.7117	**0.00**
Configuração	1	0.4698	8	0.0122	38.5753	**0.03**
Lista	3	0.0324	24	0.0038	8.6124	**0.05**
Instância * Configuração	8	0.0122	648	0.0008	14.7089	**0.00**
Instância * Lista	24	0.0038	648	0.0008	4.5376	**0.00**
Configuração * Lista	3	0.0326	24	0.0028	11.7795	**0.01**
Instância * Configuração * Lista	24	0.0028	648	0.0008	3.3456	**0.00**

Tabela 7.23 PSA com lista de candidatos - Tabela ANOVA para D1

Fonte	GL Fonte	DQM Fonte	GL Erro	DQM Erro	F	Valor de prova (%)
Instância	8	6.76E+06	648	7.47E+04	90.5106	**0.00**
Configuração	1	2.13E+07	8	1.65E+06	12.9035	**0.71**
Lista	3	4.27E+06	24	4.04E+05	10.5863	**0.01**
Instância * Configuração	8	1.65E+06	648	7.47E+04	22.1042	**0.00**
Instância * Lista	24	4.04E+05	648	7.47E+04	5.4074	**0.00**
Configuração * Lista	3	4.59E+05	24	1.39E+05	3.2958	**3.77**
Instância * Configuração * Lista	24	1.39E+05	648	7.47E+04	1.8634	**0.77**

Tabela 7.24 PSA com lista de candidatos - Tabela ANOVA para o tempo de execução

Quer para a distância D1, quer para o tempo de execução, todos os factores principais e interacções são significativos.

No diagrama de médias de D1 da Figura 7.34 observa-se que o desempenho da configuração A, em termos de qualidade de aproximação, não é afectado pela presença da lista de forma significativa. A configuração B, pelo contrário, é afectada de forma positiva. Uma análise *post hoc*, com testes de Tukey, permite concluir que o grupo de algoritmos de maior qualidade, e sem diferenças significativas entre eles em todas as instâncias, é composto apenas pelos algoritmos com base na configuração A. No entanto, o algoritmo (B, 10) está ausente deste conjunto em apenas uma instância. Graficamente (Figura 7.36), não são assinaláveis alterações de dispersão para qualquer das configurações.

No diagrama de médias do tempo de execução (Figura 7.35), é observável o aumento deste com o aumento da dimensão da *Lista*, aumento este ligeiramente superior para a configuração A. Uma análise *post hoc*, com testes de Tukey, permite concluir que o grupo de algoritmos com menores tempos de execução, e sem diferenças significativas entre eles em todas as instâncias, é composto por todos os algoritmos baseados na configuração B. O algoritmo (A, 1) está ausente deste conjunto em apenas uma instância. Tal como nos testes anteriores relativos aos tempos de execução, o diagrama do tipo caixa da Figura 7.37 exibe um aumento da dispersão com o aumento dos tempos de execução médios.

Os compromissos qualidade de aproximação / tempo de execução mais interessantes serão as configurações (A, 1) e (B, 10). Para os restantes algoritmos baseados em A não há benefícios significativos como contrapartida do aumento do tempo de execução. Relativamente à configuração base B, que apresenta os tempos de execução mais reduzidos, apenas para o caso com lista de dimensão 10, a qualidade das aproximações obtidas se aproxima dos valores obtidos com a configuração base A.

A consideração de listas de candidatos em PSA com subvizinhanças, para as configurações e tipos de instâncias aqui analisados, não conduziu à obtenção de resultados de qualidade significativamente superior à melhor configuração base original. Permitiu, no entanto, colocar o desempenho da pior configuração ao nível da melhor, em ambas as perspectivas consideradas - qualidade e tempo de execução. Tal como no caso anterior, este tipo de estratégia permite incrementar a robustez dos algoritmos em relação aos elementos das configurações base, tendo como contrapartida o incremento dos tempos de execução.

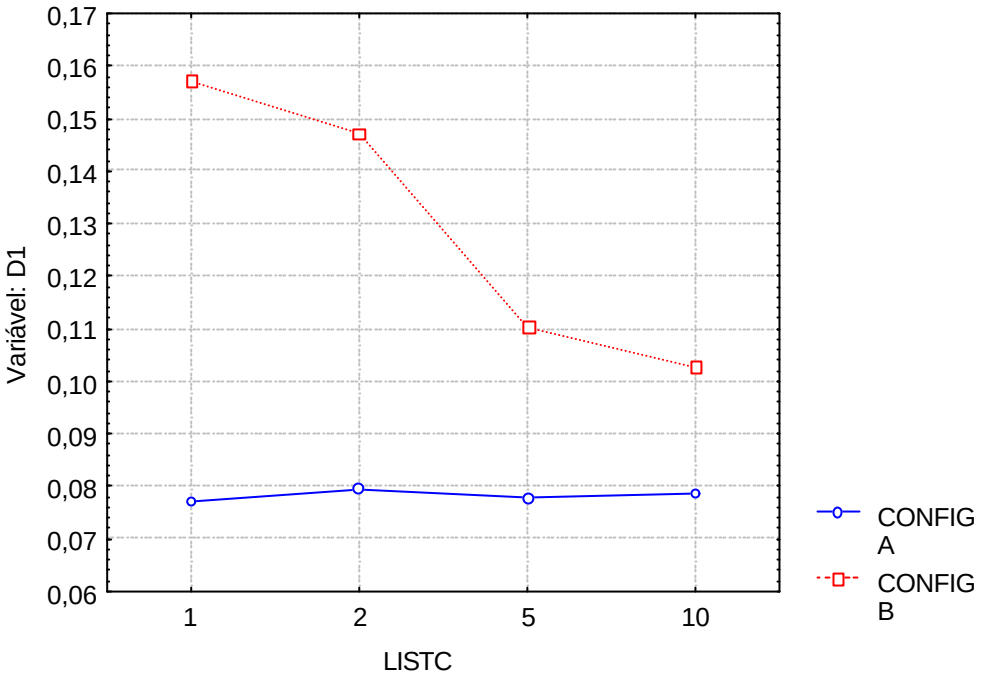


Figura 7.34 PSA com lista de candidatos - Diagrama de médias de D1 para *Configuração * Lista*

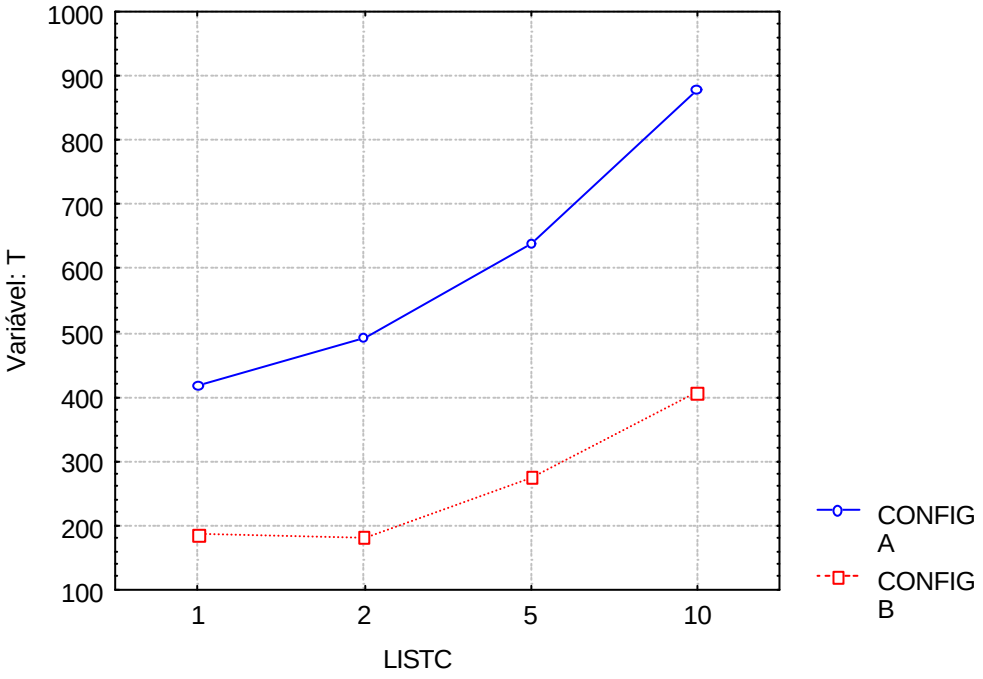


Figura 7.35 PSA com lista de candidatos - Diagrama de médias do tempo de execução para *Configuração * Lista*

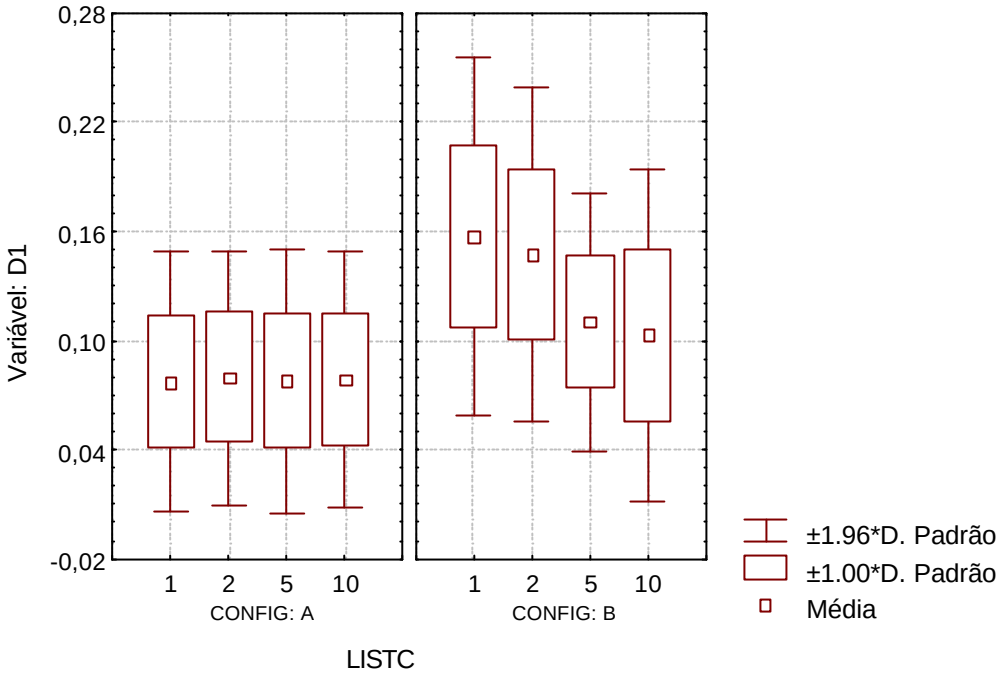


Figura 7.36 PSA com lista de candidatos - Diagrama do tipo caixa de D1 para *Configuração * Lista*

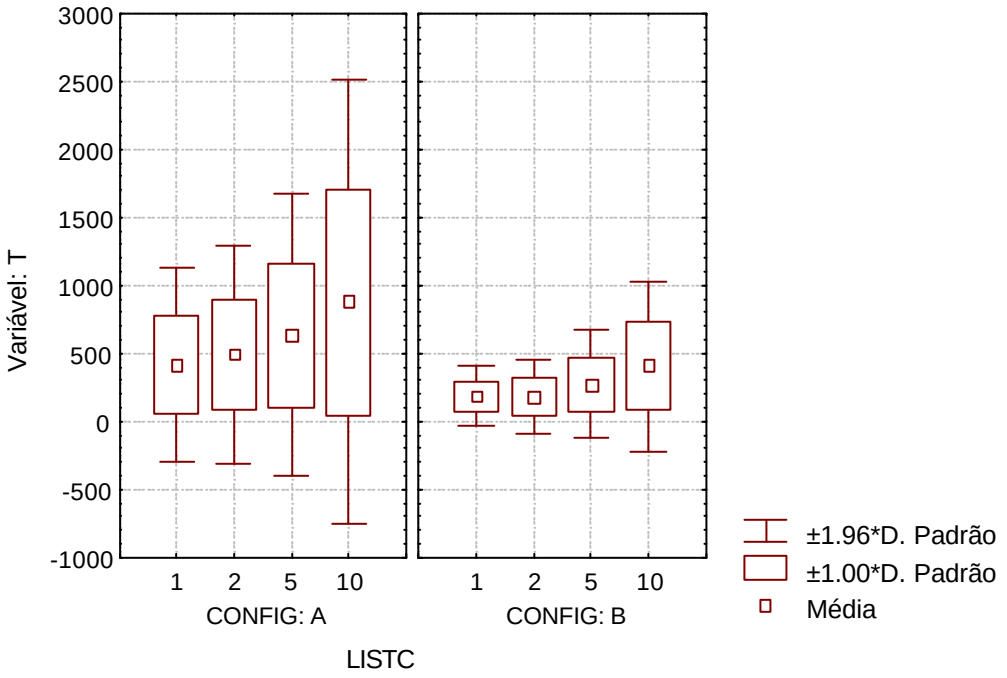


Figura 7.37 PSA com lista de candidatos - Diagrama do tipo caixa do tempo de execução para *Configuração * Lista*

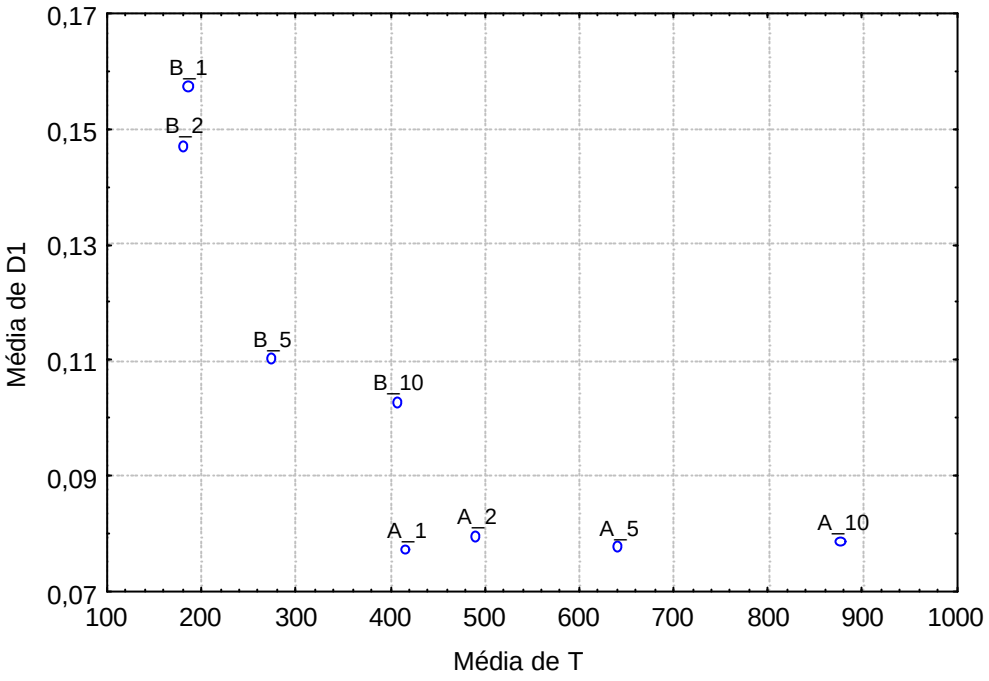


Figura 7.38 PSA com lista de candidatos - Diagrama de qualidade de aproximação e tempo de execução por algoritmo

Configuração	Dimensão da lista de candidatos	D1				T			
		Média	Desvio padrão	Nº de instâncias em que é signif. melhor		Média (s)	Desvio padrão (s)	Nº de instâncias em que é signif. menor	
				Global	Config.			Global	Config.
A	1	0.077	0.036	**9**	**9**	415.81	362.32	8	**9**
A	2	0.080	0.036	**9**	**9**	490.78	408.53	6	**9**
A	5	0.078	0.037	**9**	**9**	639.70	527.76	4	8
A	10	0.078	0.036	**9**	**9**	877.44	833.61	4	5
B	1	0.157	0.050	2	4	185.36	113.41	**9**	**9**
B	2	0.147	0.047	2	4	181.62	136.81	**9**	**9**
B	5	0.110	0.036	5	**9**	273.87	202.35	**9**	**9**
B	10	0.103	0.047	8	**9**	407.00	319.03	**9**	**9**

Tabela 7.25 PSA com lista de candidatos - Desempenho global

7.6 MOTS* com lista de candidatos

O conjunto de testes relativos às configurações de MOTS* com lista de candidatos será realizado, de acordo com o referido em 7.1.5, sobre as configurações básicas de MOTS* que apresentam o melhor e o pior desempenhos médios, ao nível da qualidade de aproximação, ou seja, respectivamente (A, 1), com (16, 7, 10), e (B, 1), com (8, 15, 5).

As tabelas ANOVA para os valores de D1 e do tempo de execução obtidos com as várias configurações de MOTS* com lista de candidatos são apresentadas, respectivamente, na Tabela 7.26 e na Tabela 7.27.

Fonte	GL Fonte	DQM Fonte	GL Erro	DQM Erro	F	Valor de prova (%)
Instância	8	0.02974	648	0.0006	50.7799	**0.00**
Configuração	1	0.34858	8	0.0156	22.3498	**0.15**
Lista	3	0.04322	24	0.0037	11.7865	**0.01**
Instância * Configuração	8	0.01560	648	0.0006	26.6284	**0.00**
Instância * Lista	24	0.00367	648	0.0006	6.2613	**0.00**
Configuração * Lista	3	0.13547	24	0.0048	28.1100	**0.00**
Instância * Configuração * Lista	24	0.00482	648	0.0006	8.2281	**0.00**

Tabela 7.26 MOTS* com lista de candidatos - Tabela ANOVA para D1

Fonte	GL Fonte	DQM Fonte	GL Erro	DQM Erro	F	Valor de prova (%)
Instância	8	5.75E+06	648	4.87E+04	117.9613	**0.00**
Configuração	1	9.40E+06	8	4.85E+05	19.3960	**0.23**
Lista	3	4.20E+05	24	1.45E+05	2.8970	*5.59*
Instância * Configuração	8	4.85E+05	648	4.87E+04	9.9421	**0.00**
Instância * Lista	24	1.45E+05	648	4.87E+04	2.9724	**0.00**
Configuração * Lista	3	1.12E+06	24	2.43E+05	4.5914	**1.12**
Instância * Configuração * Lista	24	2.43E+05	648	4.87E+04	4.9869	**0.00**

Tabela 7.27 MOTS* com lista de candidatos - Tabela ANOVA para o tempo de execução

Para a distância D1 e para o tempo de execução, todos os factores principais e interacções são significativos. No caso do tempo de execução, há a destacar o factor principal *Lista* que apresenta um valor de prova extremamente perto do limiar estabelecido para este estudo, tendo como tal sido investigado como significativo.

No diagrama de médias de D1 da Figura 7.40, é aparente uma degradação significativa da qualidade de aproximação para a configuração A, com a introdução e o crescimento da lista de candidatos. Com efeito, uma análise *post hoc*, com testes de Tukey, permite concluir que o conjunto de algoritmos de maior qualidade, e sem diferenças significativas entre eles em todas as instâncias, é composto apenas pelos algoritmos (A, 1) e (A, 2). Relativamente à configuração B, a melhor qualidade de aproximação é obtida com o algoritmo (B, 10), ausente daquele conjunto em apenas duas instâncias. Regista-se, portanto, para esta configuração, um efeito de melhoria significativa da qualidade das aproximações com a introdução e o crescimento da lista. Não são observáveis, no diagrama do tipo caixa da Figura 7.41, alterações muito relevantes dos níveis de dispersão de D1, para ambas as configurações.

No diagrama de médias do tempo de execução (Figura 7.40), é observável, para a configuração B, o respectivo aumento com o aumento da dimensão da *Lista*. A respeito da configuração A, assinala-se o desnível na passagem de uma dimensão de 2 para uma dimensão de 5 da lista de candidatos, valor próximo do qual esta deverá começar a ter um efeito mais significativo. A partir deste valor, observa-se, igualmente, um andamento muito semelhante das duas configurações, para a qualidade da aproximação e os tempos de execução. Uma análise *post hoc*, com testes de Tukey, permite concluir que o grupo de algoritmos com menores tempos de execução, e sem diferenças significativas entre eles em todas as instâncias, é composto por (B, 1), (B, 2) e (A, 5). (B, 5) e (B, 10) estão ausentes deste conjunto em apenas uma e duas instâncias, respectivamente. O aumento da dispersão dos tempos de execução, com o aumento dos respectivos valores médios, é, uma vez mais, constatável no diagrama do tipo caixa da Figura 7.42.

Não surge naturalmente destacada qualquer configuração, de entre as analisadas, no que respeita aos compromissos qualidade de aproximação / tempo. A lista de candidatos tem um efeito de aproximação das duas configurações, com uma redução simultânea da qualidade e do tempo de execução, no caso da configuração A, e o inverso, no caso da configuração B.

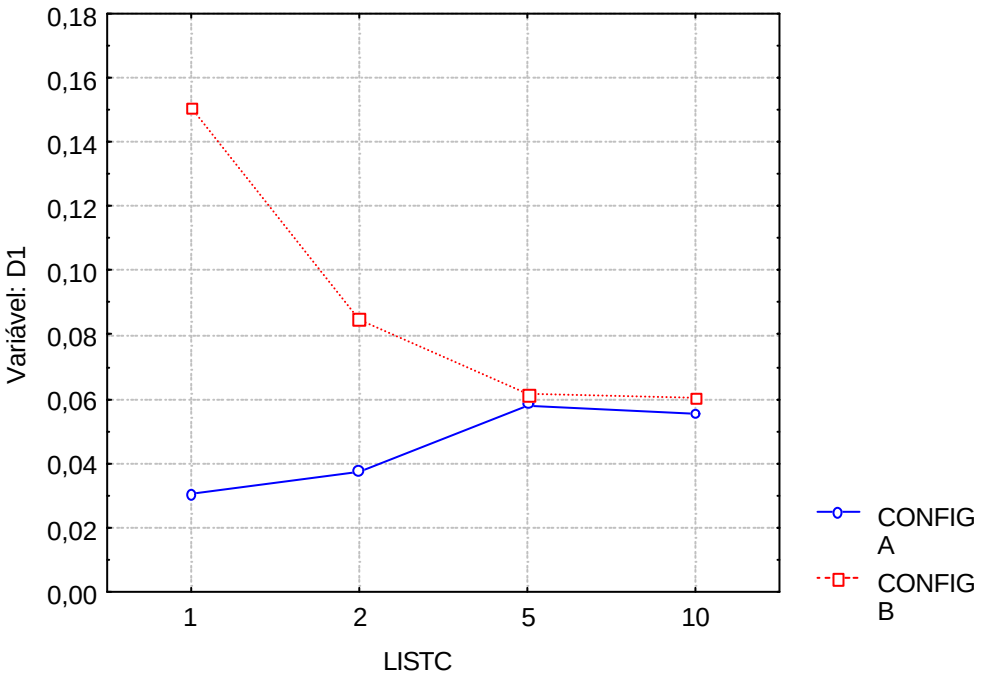


Figura 7.39 MOTS* com lista de candidatos - Diagrama de médias de D1 para *Configuração* * *Lista*

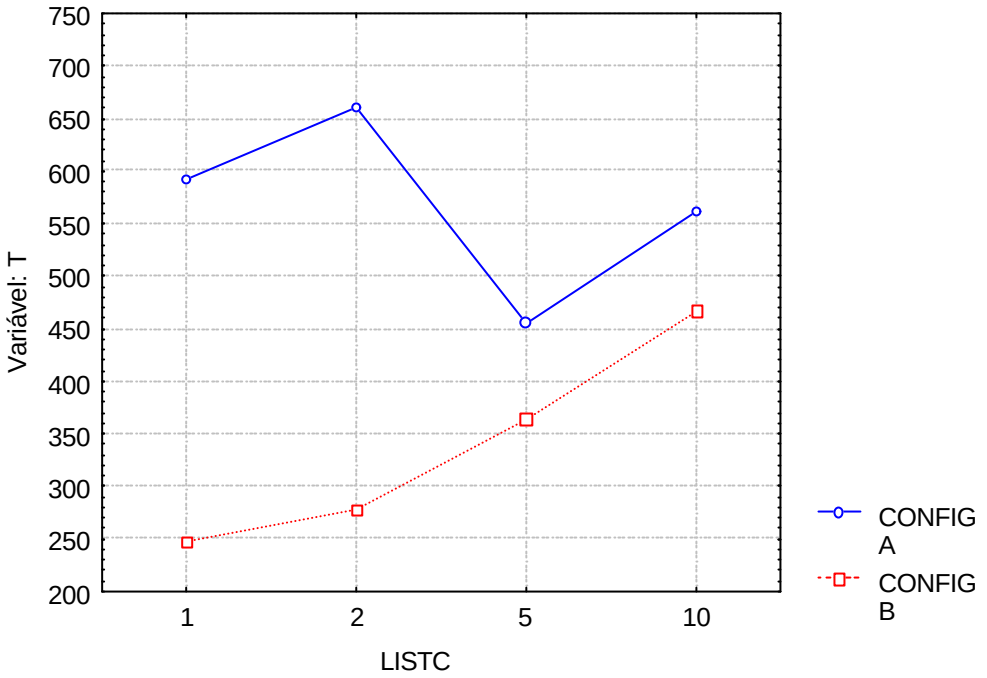


Figura 7.40 MOTS* com lista de candidatos - Diagrama de médias do tempo de execução para *Configuração * Lista*

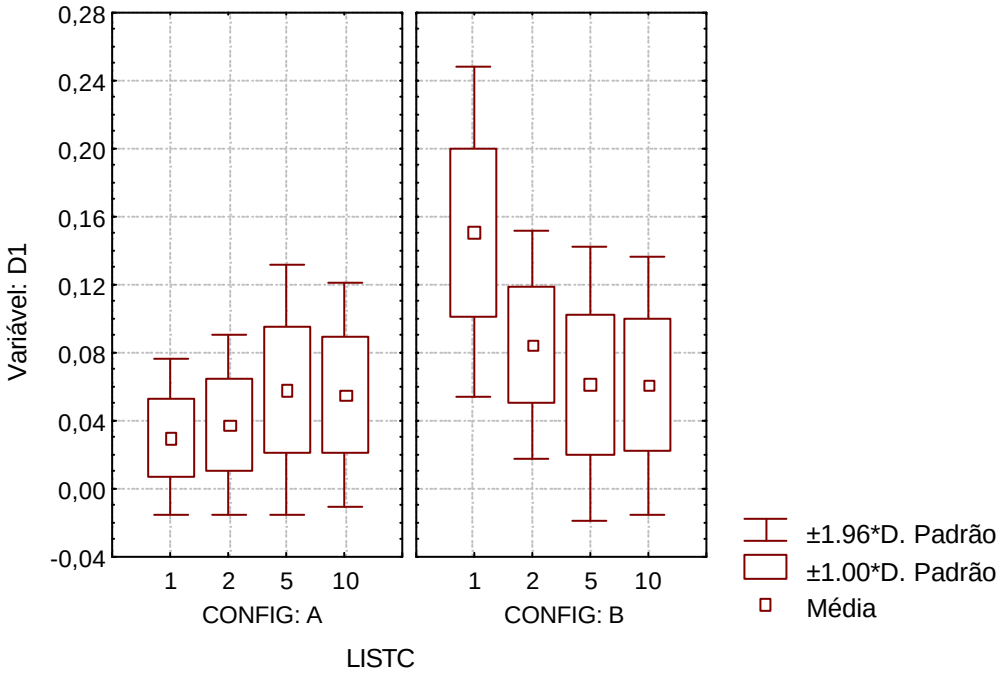


Figura 7.41 MOTS* com lista de candidatos - Diagrama do tipo caixa de D1 para *Configuração * Lista*

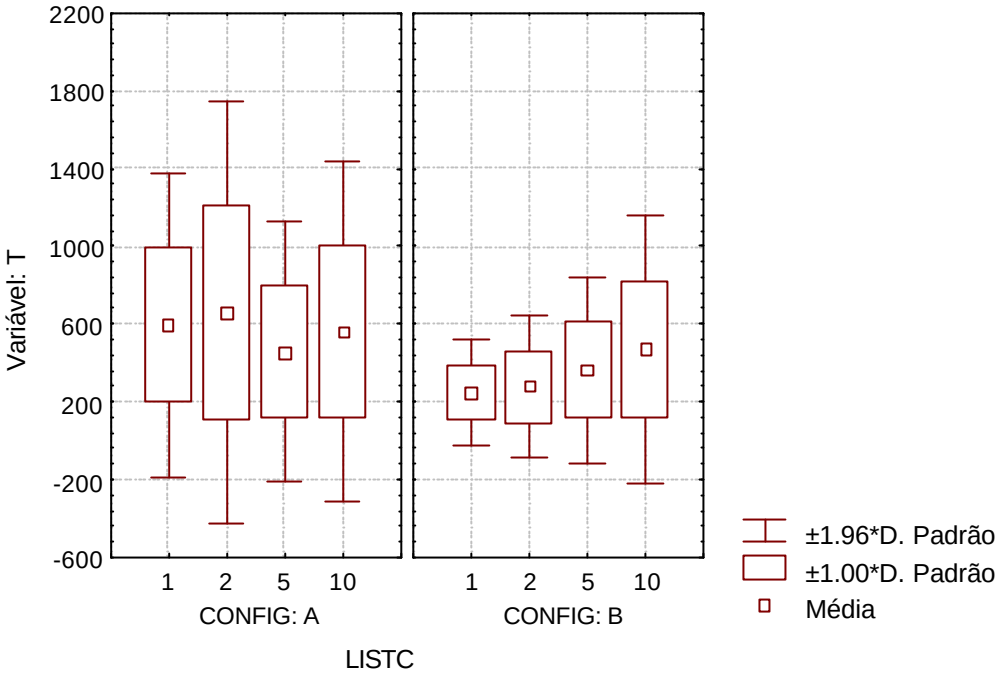


Figura 7.42 MOTS* com lista de candidatos - Diagrama do tipo caixa do tempo de execução para *Configuração * Lista*

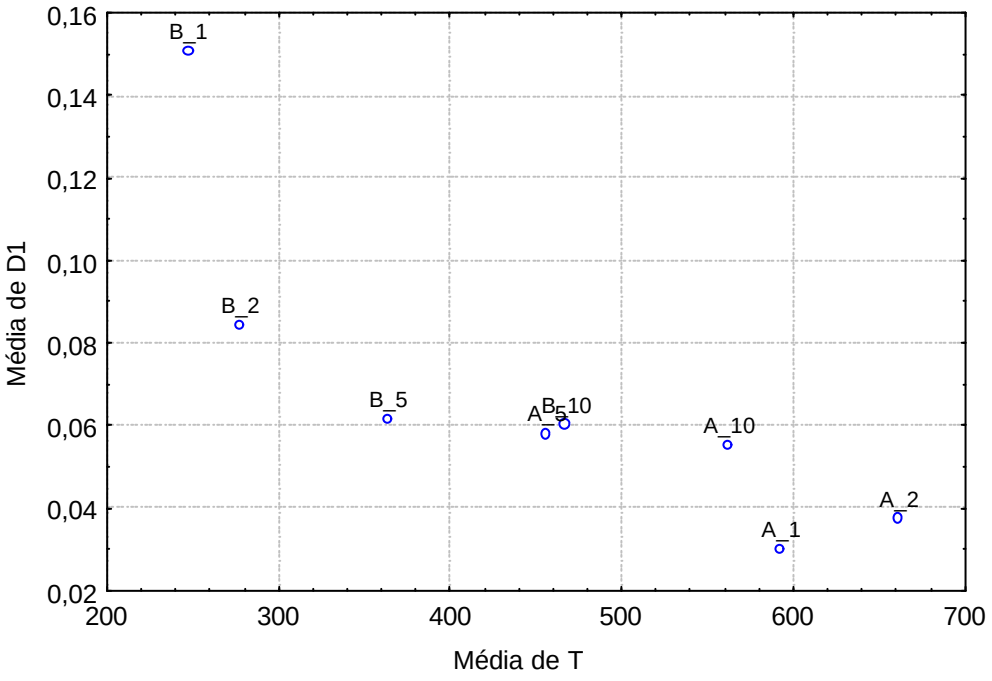


Figura 7.43 MOTS* com lista de candidatos - Diagrama de qualidade de aproximação e tempo de execução por algoritmo

Configuração	Dimensão da lista de candidatos	D1				T			
		Média	Desvio padrão	Nº de instâncias em que é signif. melhor		Média (s)	Desvio padrão (s)	Nº de instâncias em que é signif. menor	
				Global	Config.			Global	Config.
A	1	0.030	0.023	**9**	**9**	591.91	397.02	4	8
A	2	0.038	0.027	**9**	**9**	660.84	553.75	5	7
A	5	0.058	0.037	6	6	455.45	341.82	**9**	**9**
A	10	0.055	0.034	6	6	561.22	447.75	5	9
B	1	0.151	0.049	1	2	247.07	137.52	**9**	**9**
B	2	0.085	0.034	2	6	277.29	187.26	**9**	**9**
B	5	0.061	0.041	6	**9**	363.95	243.96	8	8
B	10	0.060	0.039	7	**9**	467.08	353.20	7	7

Tabela 7.28 MOTS* com lista de candidatos - Desempenho global

7.7 PSA com vizinhança variável

Conforme referido em 7.1.5, este conjunto de testes será conduzido sobre as configurações básicas de PSA que apresentam o melhor e o pior desempenhos médios, ao nível da qualidade de aproximação, ou seja, respectivamente (A, N), com (16, 0.005, 5), e (B, N), com (8, 2, 200).

As tabelas ANOVA para os valores de D1 e do tempo de execução obtidos com as várias configurações de PSA com vizinhança variável são apresentadas, respectivamente, na Tabela 7.29 e na Tabela 7.30.

Fonte	GL Fonte	DQM Fonte	GL Erro	DQM Erro	F	Valor de prova (%)
Instância	8	0.0764	324	0.0092	8.3470	**0.00**
Configuração	1	1.4898	8	0.0596	25.0166	**0.11**
Vizinhança	1	0.0748	8	0.0072	10.3315	**1.23**
Instância * Configuração	8	0.0596	324	0.0092	6.5031	**0.00**
Instância * Vizinhança	8	0.0072	324	0.0092	0.7901	61.17
Configuração * Vizinhança	1	0.0071	8	0.0071	1.0022	34.61
Instância * Configuração * Vizinhança	8	0.0071	324	0.0092	0.7767	62.36

Tabela 7.29 PSA com vizinhança variável - Tabela ANOVA para D1

Fonte	GL Fonte	DQM Fonte	GL Erro	DQM Erro	F	Valor de prova (%)
Instância	8	1.35E+05	324	3.39E+03	39.8983	**0.00**
Configuração	1	1.85E+04	8	8.92E+03	2.0687	18.83
Vizinhança	1	2.26E+04	8	6.66E+03	3.3916	10.28
Instância * Configuração	8	8.92E+03	324	3.39E+03	2.6322	**0.83**
Instância * Vizinhança	8	6.66E+03	324	3.39E+03	1.9659	5.01
Configuração * Vizinhança	1	5.73E+03	8	5.78E+03	0.9921	34.84
Instância * Configuração * Vizinhança	8	5.78E+03	324	3.39E+03	1.7053	9.62

Tabela 7.30 PSA com vizinhança variável - Tabela ANOVA para o tempo de execução

Relativamente à qualidade de aproximação, o factor *Vizinhança* apresenta-se significativo, não possuindo, no entanto, interacções significativas com os restantes factores. Para o tempo de execução, o factor não é significativo, o mesmo sucedendo com todas as interacções que o envolvem.

Embora exista uma melhoria dos valores médios da qualidade de aproximação, acompanhada de um aumento do tempo de execução (Figura 7.44 e Figura 7.45), na análise *post-hoc*, com testes de Tukey, praticamente não se encontram variações estatisticamente significativas. Os diagramas do tipo caixa da Figura 7.46 e da Figura 7.47 suportam graficamente esta indicação. Ao nível da dispersão, não são observáveis graficamente alterações com a utilização da vizinhança variável, quer para D1, quer para os tempos de execução.

A utilização da vizinhança variável considerada não tem qualquer efeito significativo sobre o desempenho do PSA, ao nível da qualidade das aproximações ou ao nível dos tempos de execução. Embora exista, para ambas as configurações, uma melhoria da qualidade de aproximação média, acompanhada de um incremento do tempo de execução médio, estas variações não são estatisticamente significativas.

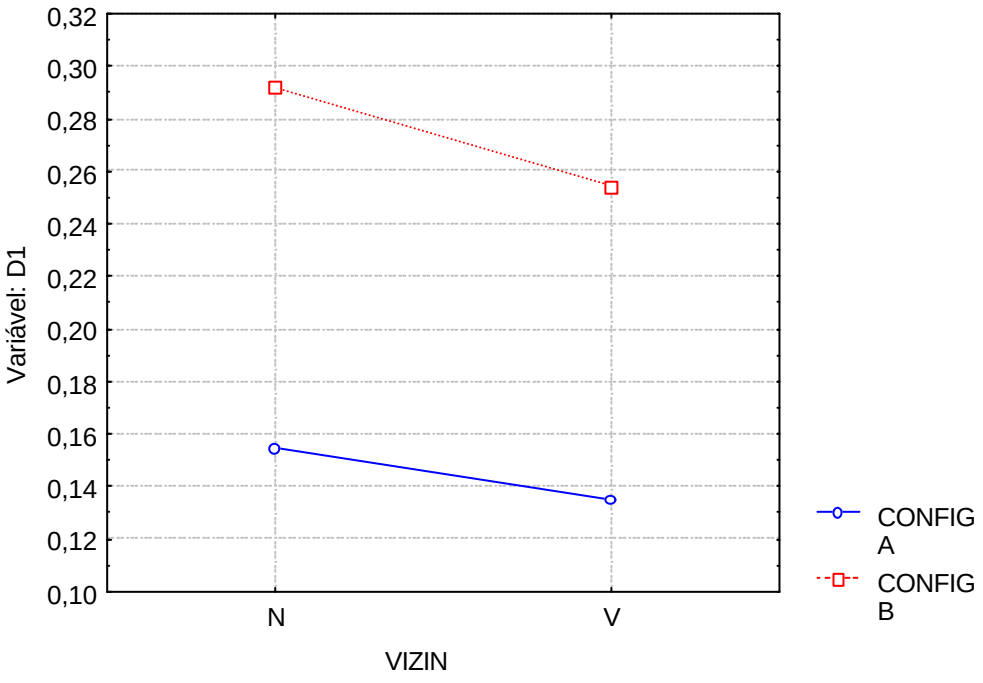


Figura 7.44 PSA com vizinhança variável - Diagrama de médias de D1 para *Configuração* * *Vizinhança*

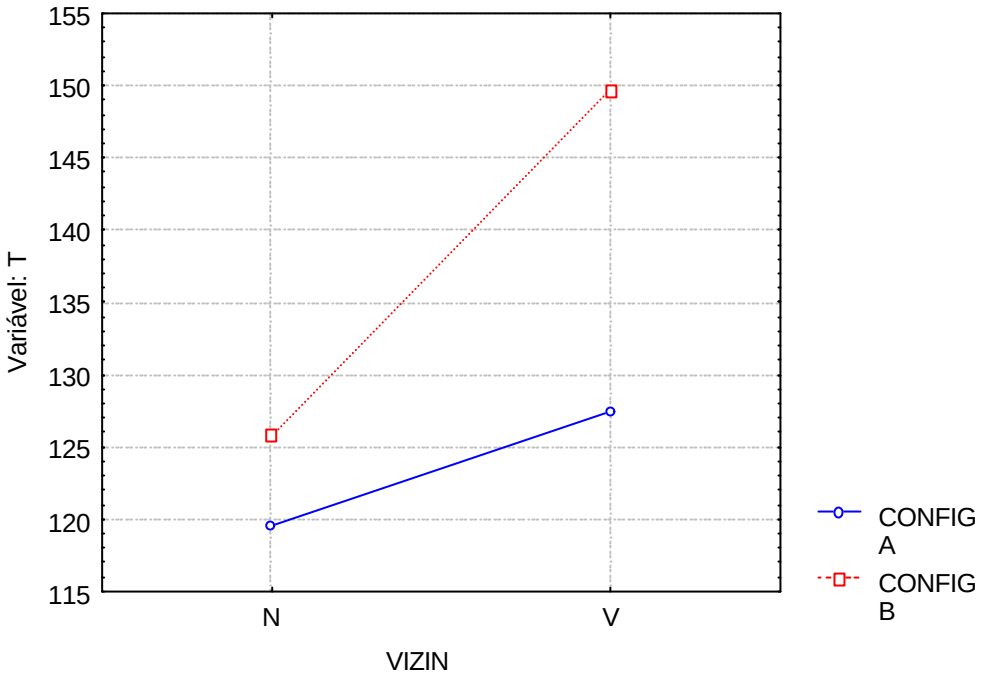


Figura 7.45 PSA com vizinhança variável - Diagrama de médias do tempo de execução para *Configuração * Vizinhança*

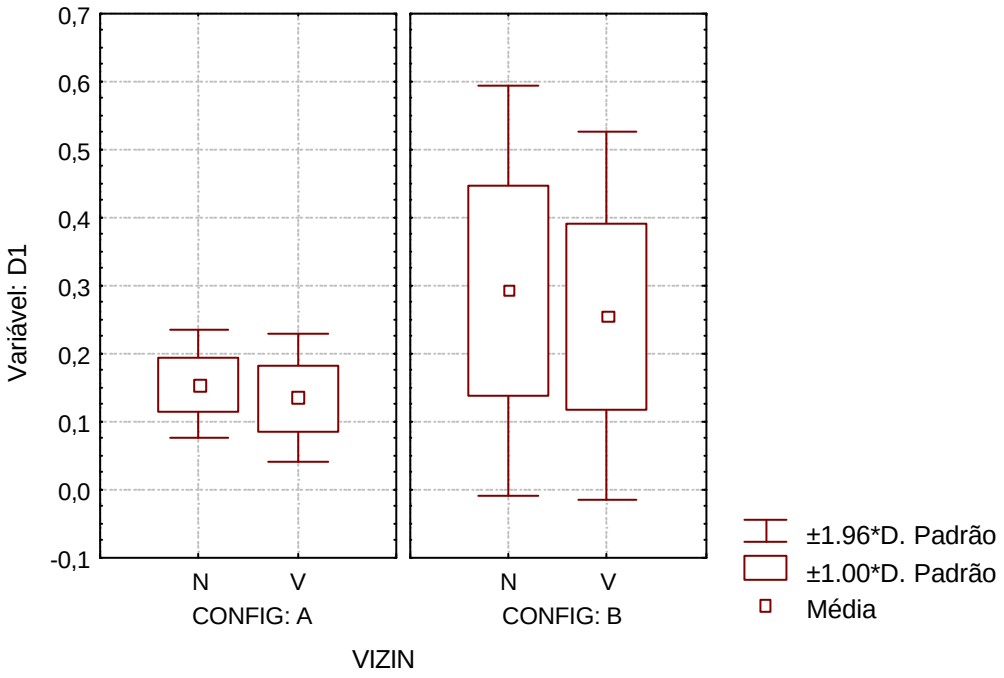


Figura 7.46 PSA com vizinhança variável - Diagrama do tipo caixa de D1 para *Configuração * Vizinhança*

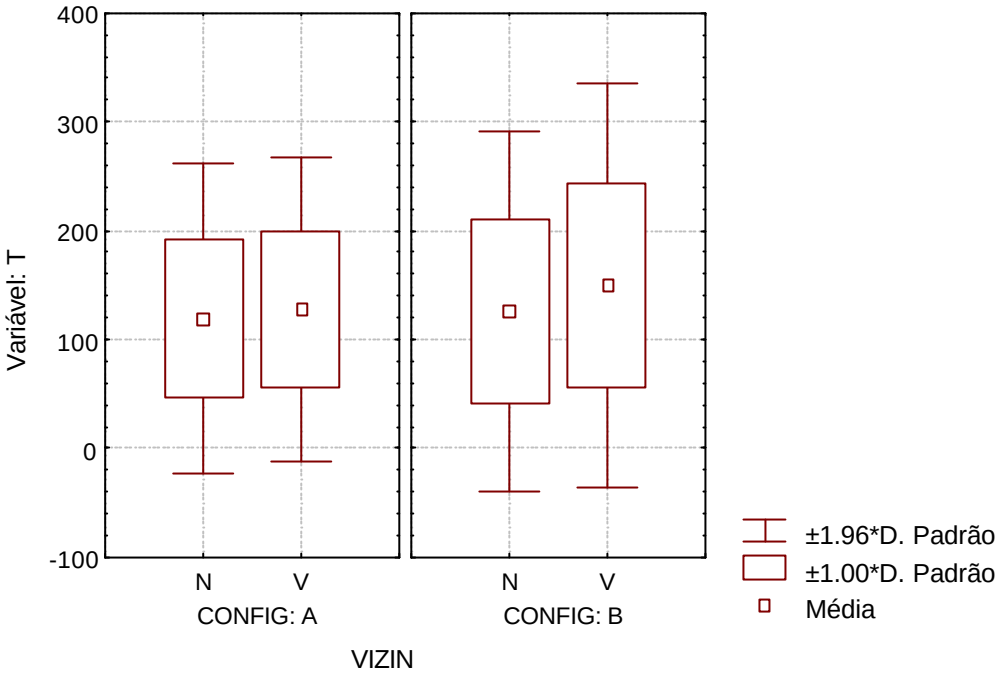


Figura 7.47 PSA com vizinhança variável - Diagrama do tipo caixa do tempo de execução para *Configuração * Vizinhança*

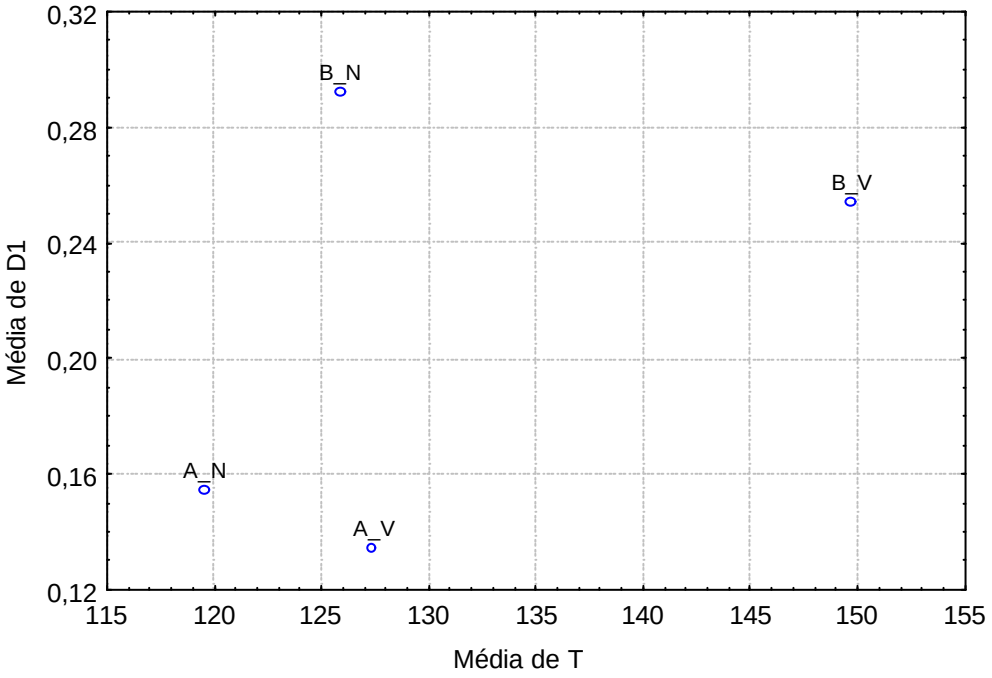


Figura 7.48 PSA com vizinhança variável - Diagrama de qualidade de aproximação e tempo de execução por algoritmo

	Tipo de vizinhança	D1				T			
		Média	Desvio padrão	Nº de instâncias em que é signif. melhor		Média (s)	Desvio padrão (s)	Nº de instâncias em que é signif. menor	
				Global	Config.			Global	Config.
A	N	0.154	0.041	**9**	**9**	119.51	72.87	**9**	**9**
A	V	0.135	0.048	**9**	**9**	127.37	71.00	8	8
B	N	0.292	0.155	6	**9**	125.84	84.61	**9**	**9**
B	V	0.254	0.138	7	**9**	149.67	94.33	8	**9**

Tabela 7.31 PSA com vizinhança variável - Desempenho global

7.8 MOTS* com vizinhança variável

O conjunto de testes relativos às configurações de MOTS* com lista de candidatos será realizado, de acordo com o referido em 7.1.5, sobre as configurações básicas de MOTS* que apresentam o melhor e o pior desempenhos médios, ao nível da qualidade de aproximação, ou seja, respectivamente (A, N), com (16, 7, 10), e (B, N), com (8, 15, 5).

As tabelas ANOVA para os valores de D1 e do tempo de execução obtidos com as várias configurações de MOTS* com vizinhança variável são apresentadas, respectivamente, na Tabela 7.32 e na Tabela 7.33.

Para a distância D1, todos os factores principais e interacções são significativos. Para o tempo de execução, apenas o factor *Vizinhança*, individualmente, não é significativo.

Fonte	GL Fonte	DQM Fonte	GL Erro	DQM Erro	F	Valor de prova (%)
Instância	8	0.0233	324	0.0010	23.7355	**0.00**
Configuração	1	4.4751	8	0.0293	152.4809	**0.00**
Vizinhança	1	1.1399	8	0.0185	61.6371	**0.00**
Instância * Configuração	8	0.0293	324	0.0010	29.8903	**0.00**
Instância * Vizinhança	8	0.0185	324	0.0010	18.8352	**0.00**
Configuração * Vizinhança	1	0.9444	8	0.0184	51.4400	**0.01**
Instância * Configuração * Vizinhança	8	0.0184	324	0.0010	18.6971	**0.00**

Tabela 7.32 MOTS* com vizinhança variável - Tabela ANOVA para D1

Fonte	GL Fonte	DQM Fonte	GL Erro	DQM Erro	F	Valor de prova (%)
Instância	8	2.50E+06	324	4.06E+04	61.4116	**0.00**
Configuração	1	2.28E+07	8	1.48E+06	15.3556	**0.44**
Vizinhança	1	2.61E+05	8	1.00E+05	2.6077	14.50
Instância * Configuração	8	1.48E+06	324	4.06E+04	36.4878	**0.00**
Instância * Vizinhança	8	1.00E+05	324	4.06E+04	2.4599	**1.34**
Configuração * Vizinhança	1	2.25E+06	8	1.49E+05	15.0692	**0.47**
Instância * Configuração * Vizinhança	8	1.49E+05	324	4.06E+04	3.6757	**0.04**

Tabela 7.33 MOTS* com vizinhança variável - Tabela ANOVA para o tempo de execução

No diagrama de médias de D1 da Figura 7.49 observa-se que a configuração A, ao nível da qualidade de aproximação, não é afectada de forma significativa pelo tipo de vizinhança utilizado. Ao invés, a configuração B é-o, de forma negativa. Uma análise *post hoc*, com testes de Tukey, permite concluir que o conjunto de algoritmos de maior qualidade, e sem diferenças significativas entre eles em todas as instâncias, se compõe apenas de (A, N), com (A, V) fora deste conjunto em apenas uma instância. Esta mesma análise permite ainda verificar que (B, N) é significativamente melhor que (B, V) para todas as instâncias. Graficamente (Figura 7.51), verificam-se incrementos de dispersão da qualidade de aproximação, em ambas as configurações, com a utilização da vizinhança variável.

No diagrama de médias do tempo de execução (Figura 7.50), é observável, para a configuração B, uma ligeira redução dos valores médios com a utilização da vizinhança variável, enquanto para a configuração A se regista um ligeiro acréscimo. Uma análise *post hoc*, com testes de Tukey, revela que não há diferenças significativas para a configuração B, em todas as instâncias, enquanto que para a configuração A tal sucede em 6 instâncias, apresentando (A, V) valores significativamente mais elevados do que (A, N) nas restantes 3. Em 6 instâncias, a configuração A apresenta tempos de execução significativamente maiores do que B. No diagrama do tipo caixa da Figura 7.52, observa-se novamente o incremento da dispersão com o aumento dos tempos de execução.

Ao nível da qualidade das aproximações, não se registaram melhorias significativas com a utilização da vizinhança variável, tendo-se mesmo verificado uma degradação para o caso da configuração B. Ao nível do tempo de execução, não se registaram alterações significativas, dentro de cada uma das configurações consideradas.

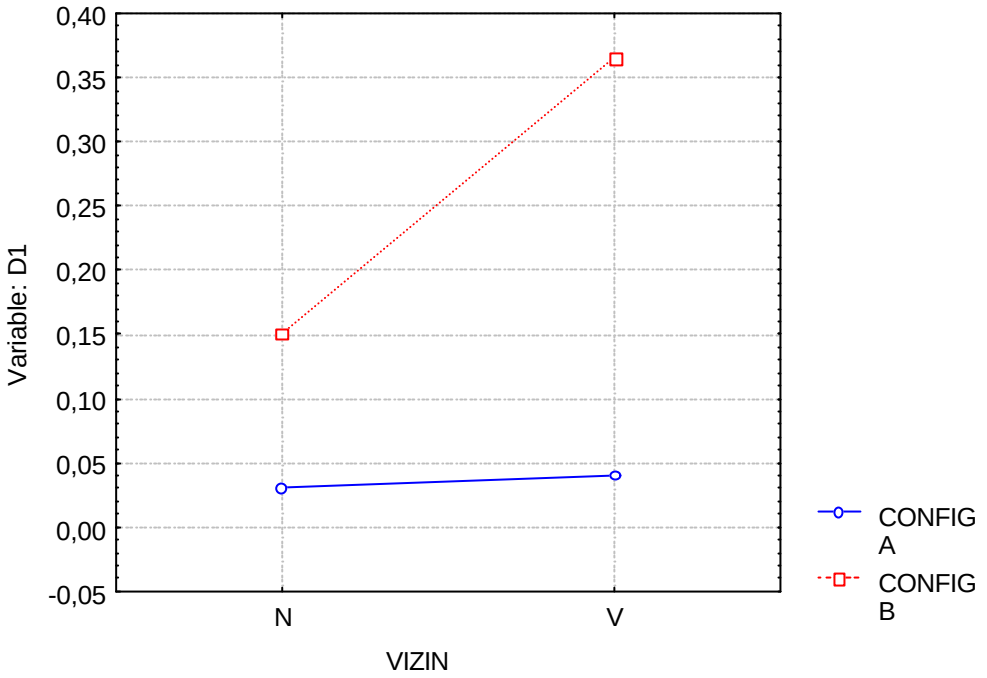


Figura 7.49 MOTS* com vizinhança variável - Diagrama de médias de D1 para *Configuração * Vizinhança*

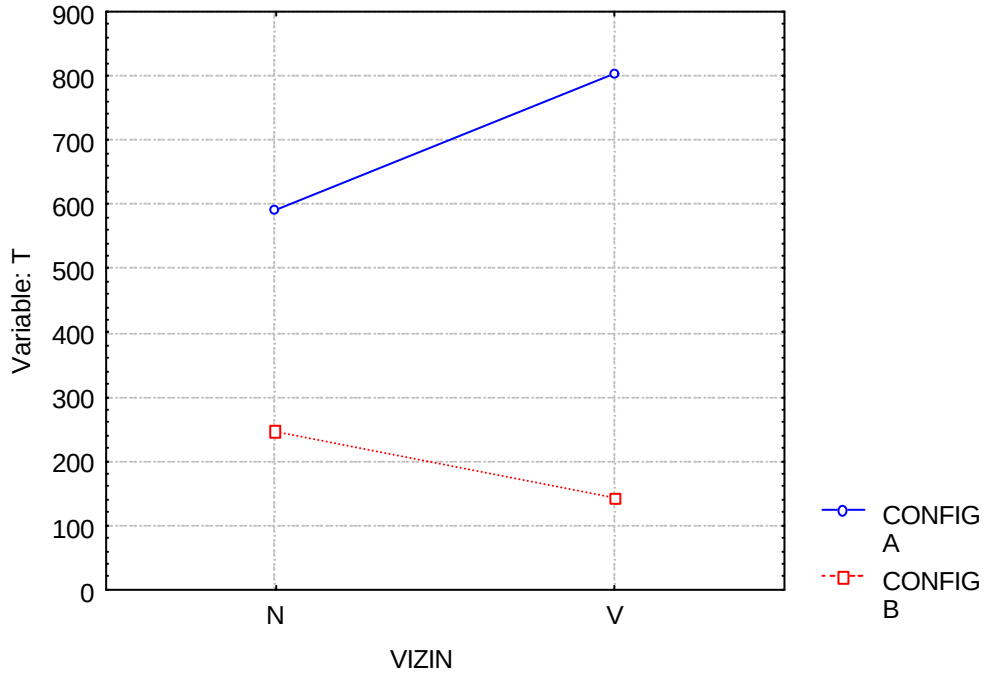


Figura 7.50 MOTS* com vizinhança variável - Diagrama de médias do tempo de execução para *Configuração * Vizinhança*

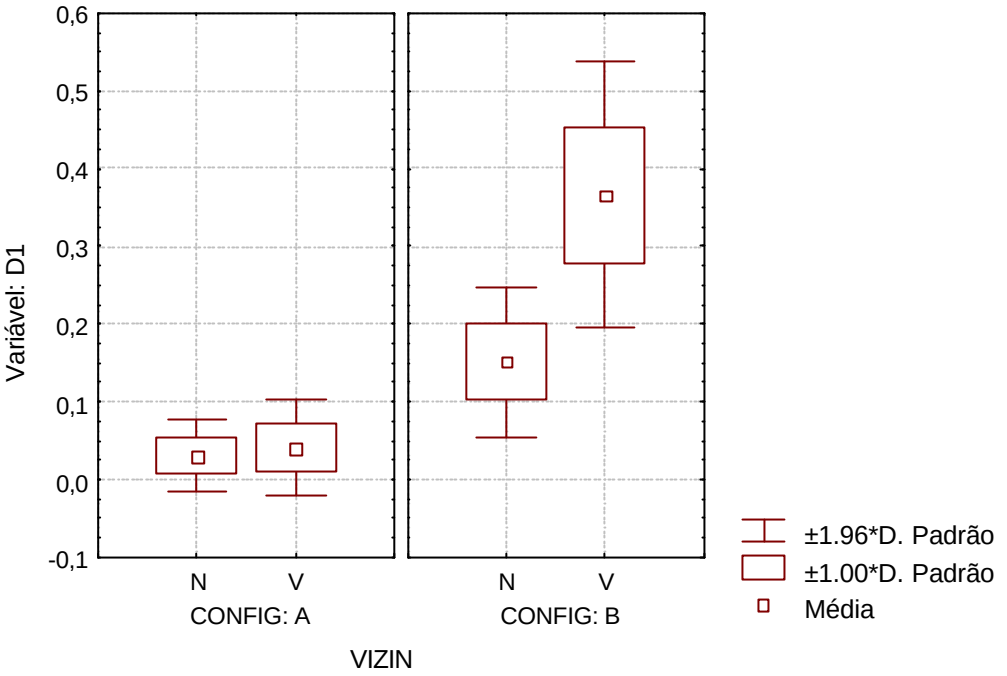


Figura 7.51 MOTS* com vizinhança variável - Diagrama do tipo caixa de D1 para *Configuração * Vizinhança*

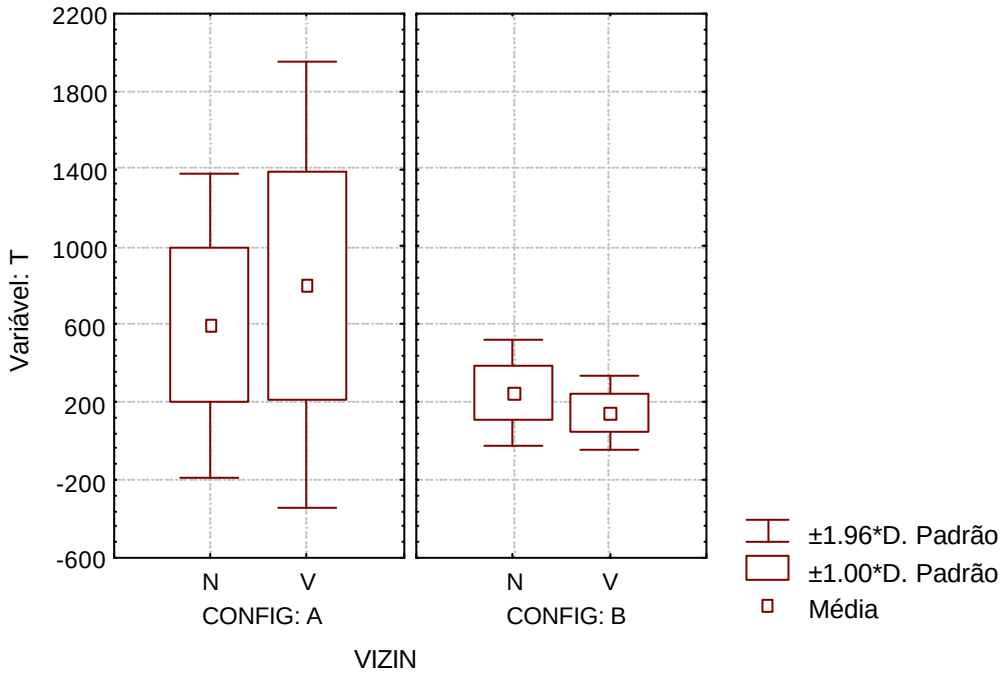


Figura 7.52 MOTS* com vizinhança variável - Diagrama do tipo caixa do tempo de execução para *Configuração * Vizinhança*

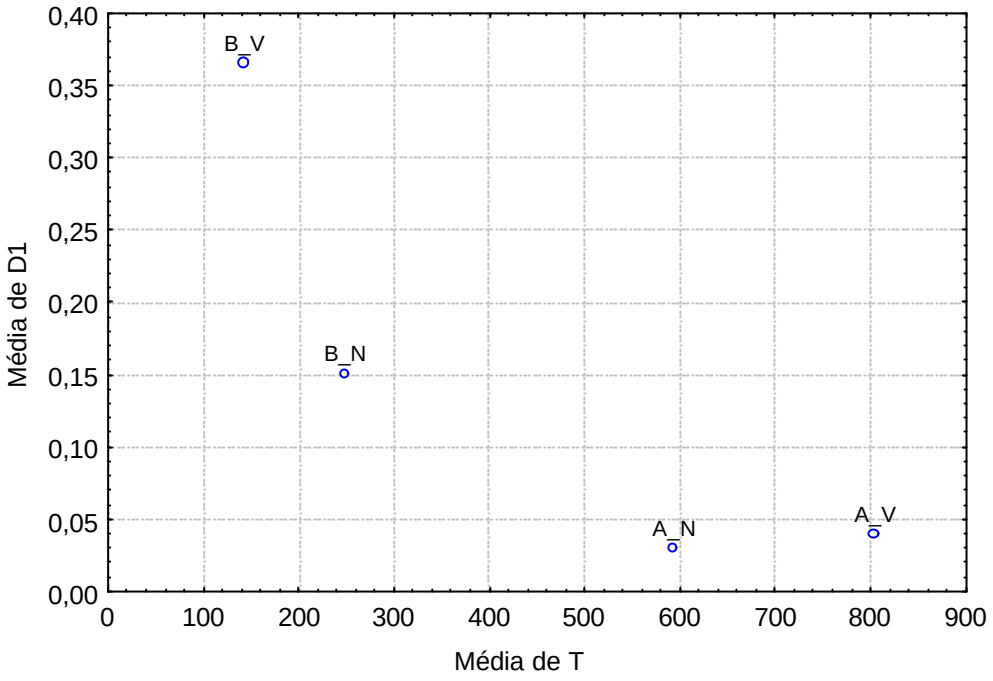


Figura 7.53 MOTS* com vizinhança variável - Diagrama de qualidade de aproximação e tempo de execução por algoritmo

Configuração	Tipo de vizinhança	D1				T			
		Média	Desvio padrão	Nº de instâncias em que é signif. melhor		Média (s)	Desvio padrão (s)	Nº de instâncias em que é signif. menor	
				Global	Config.			Global	Config.
A	N	0.030	0.023	**9**	**9**	591.91	397.02	3	**9**
A	V	0.040	0.032	8	8	803.88	584.95	3	6
B	N	0.151	0.049	1	**9**	247.07	137.52	**9**	**9**
B	V	0.366	0.087	0	0	142.74	96.33	**9**	**9**

Tabela 7.34 MOTS* com vizinhança variável - Desempenho global

7.9 Hibridização e paralelização de alto nível

Conforme indicado em 7.1.5, as configurações a avaliar a este nível, hibridizam e paralelizam pares de algoritmos PSA-MOTS*, que na execução individual apresentem idênticos níveis de qualidade de aproximação.

7.9.1 Configurações base com melhor qualidade de aproximação média

O par de algoritmos com melhor qualidade de aproximação média, que será usado na configuração híbrida e paralela de alto nível HP_A, é constituído por: TS_A, com (8, 7, 5), e SA_A, com (16, 0.005, 5) e subvizinhança 10.

As tabelas ANOVA para os valores de D1 e do tempo de execução obtidos com as 3 configurações, são apresentadas, respectivamente, na Tabela 7.35 e na Tabela 7.36.

Fonte	GL Fonte	DQM Fonte	GL Erro	DQM Erro	F	Valor de prova (%)
Instância	8	0.0149	243	0.0005	27.8986	**0.00**
Algoritmo	2	0.0121	16	0.0028	4.2994	**3.20**
Instância * Algoritmo	16	0.0028	243	0.0005	5.2967	**0.00**

Tabela 7.35 Hibridização e paralelização de alto nível (configurações base com melhor qualidade de aproximação média) - Tabela ANOVA para D1

Fonte	GL Fonte	DQM Fonte	GL Erro	DQM Erro	F	Valor de prova (%)
Instância	8	2.31E+06	243	3.02E+04	76.3524	**0.00**
Algoritmo	2	3.18E+06	16	2.92E+05	10.8938	**0.10**
Instância * Algoritmo	16	2.92E+05	243	3.02E+04	9.6579	**0.00**

Tabela 7.36 Hibridização e paralelização de alto nível (configurações base com melhor qualidade de aproximação média) - Tabela ANOVA para o tempo de execução

Todos os factores principais e interacções são significativos, quer para a qualidade das aproximações, quer para o tempo de execução.

No diagrama de médias de D1 da Figura 7.54, é aparente uma ligeira melhoria da qualidade de aproximação com a configuração híbrida e paralela de alto nível. Com efeito, uma análise *post hoc*, com testes de Tukey, permite concluir que o conjunto de algoritmos de maior qualidade, e sem diferenças significativas entre eles em todas as instâncias, é composto apenas pelo algoritmo HP_A. TS_A e SA_A ficam ausentes daquele conjunto em apenas uma e duas instâncias, respectivamente. Regista-se, portanto, para esta configuração, um efeito de melhoria significativa da qualidade das aproximações. Graficamente (Figura 7.57), a dispersão apresenta-se sensivelmente idêntica para as três configurações.

Relativamente aos tempos de execução, observa-se no diagrama de médias da Figura 7.55 a existência de uma diferença assinalável entre o algoritmo MOTS* e os restantes. Uma análise *post hoc*, com testes de Tukey, permite concluir que apenas este algoritmo apresenta

tempos de execução significativamente menores em todas as instâncias. Na comparação entre os restantes algoritmos, verifica-se que em 3 instâncias o algoritmo SA_A apresenta tempos de execução significativamente menores, enquanto nas restantes não há diferenças significativas. A dispersão aumenta com o tempo de execução médio (Figura 7.57).

A hibridização das configurações base com melhor qualidade de aproximação média permitiu obter uma ligeira melhoria da qualidade das aproximações, acompanhada por um aumento do tempo de execução.

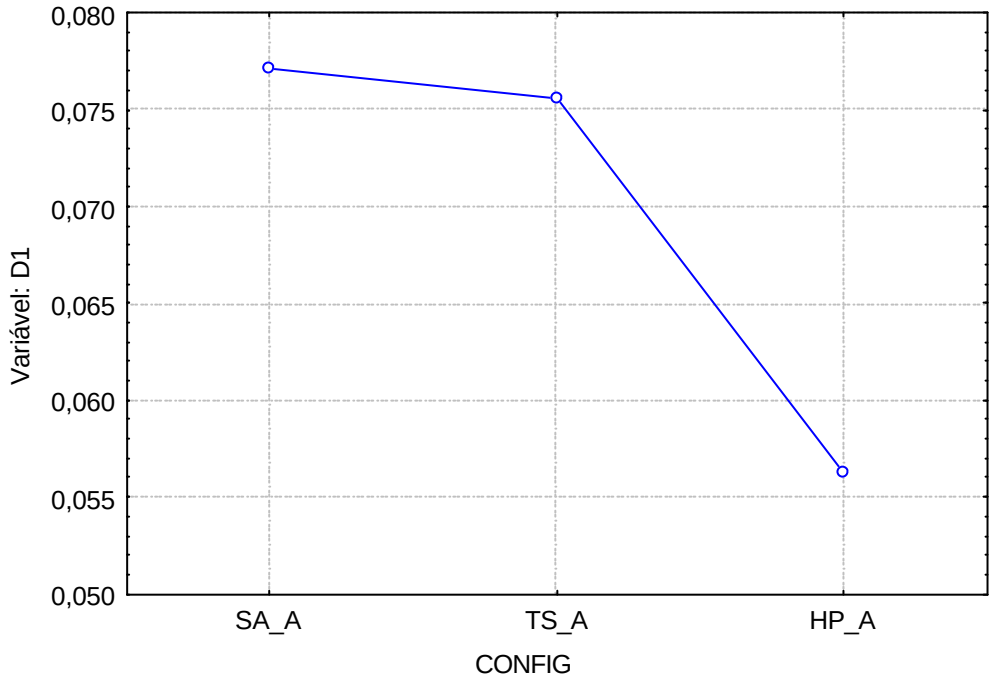


Figura 7.54 Hibridização e paralelização de alto nível (configurações base com melhor qualidade de aproximação média) - Diagrama de médias de D1 para *Algoritmo*

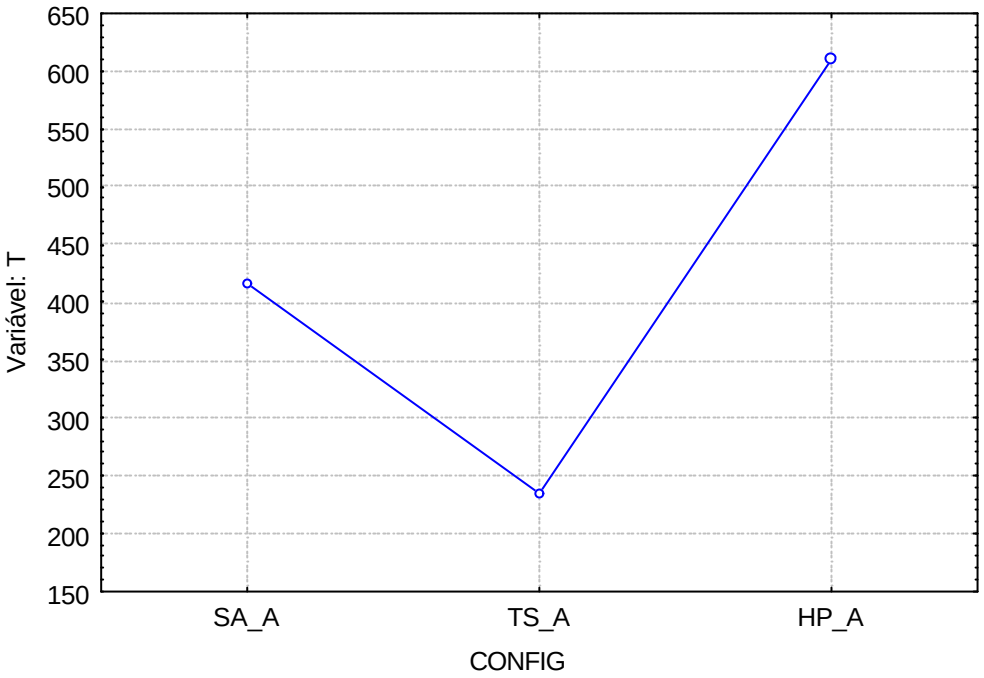


Figura 7.55 Hibridização e paralelização de alto nível (configurações base com melhor qualidade de aproximação média) - Diagrama de médias do tempo de execução para *Algoritmo*

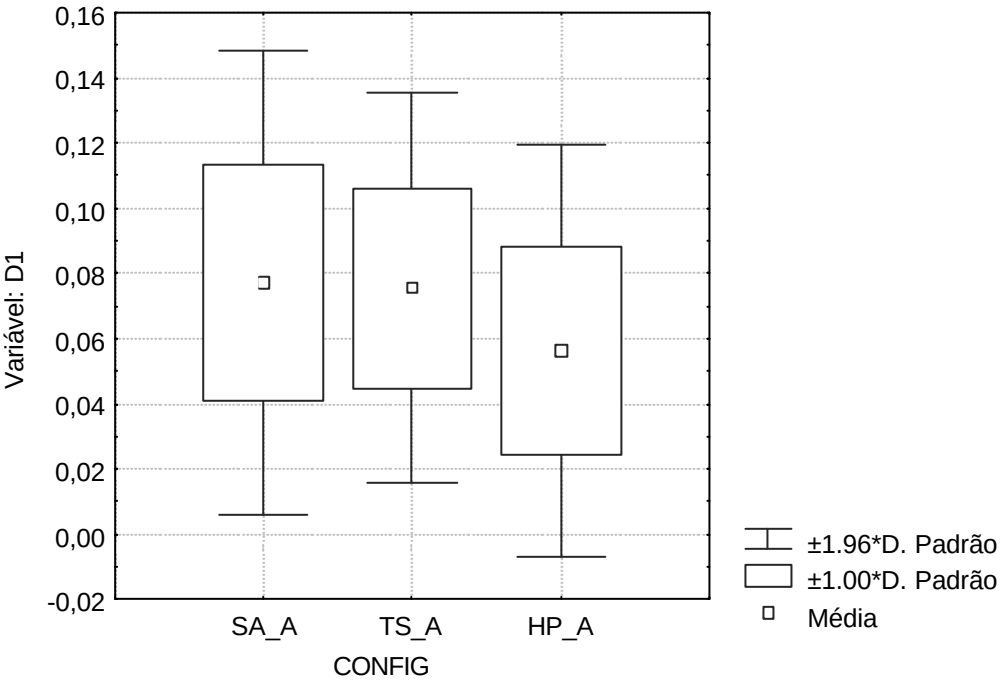


Figura 7.56 Hibridização e paralelização de alto nível (configurações base com melhor qualidade de aproximação média) - Diagrama do tipo caixa de D1 para *Algoritmo*

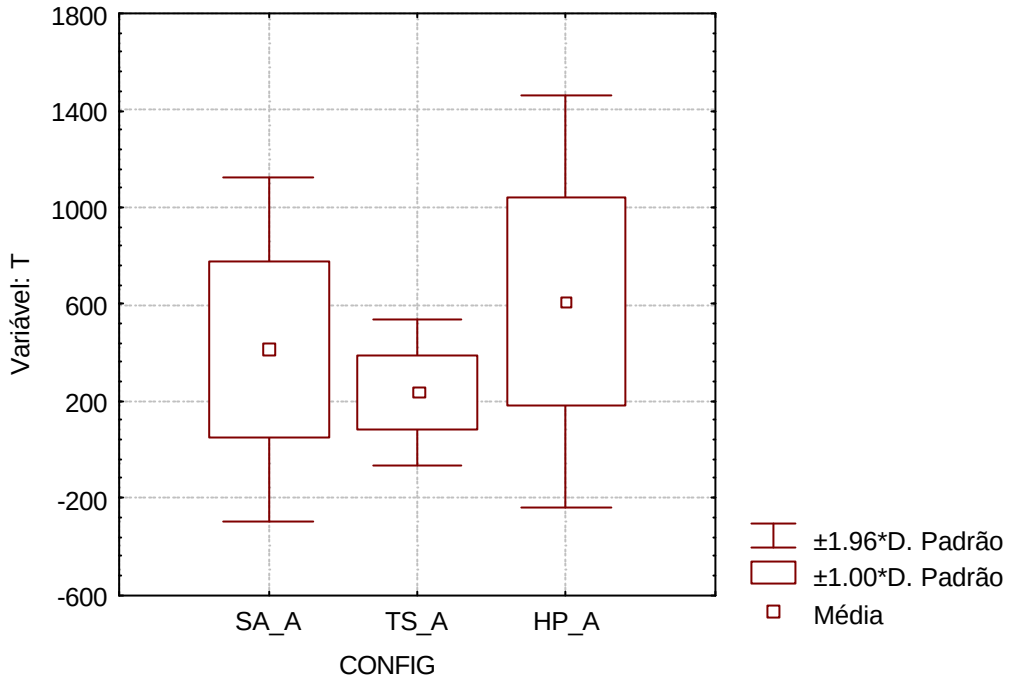


Figura 7.57 Hibridização e paralelização de alto nível (configurações base com melhor qualidade de aproximação média) - Diagrama do tipo caixa do tempo de execução para *Algoritmo*

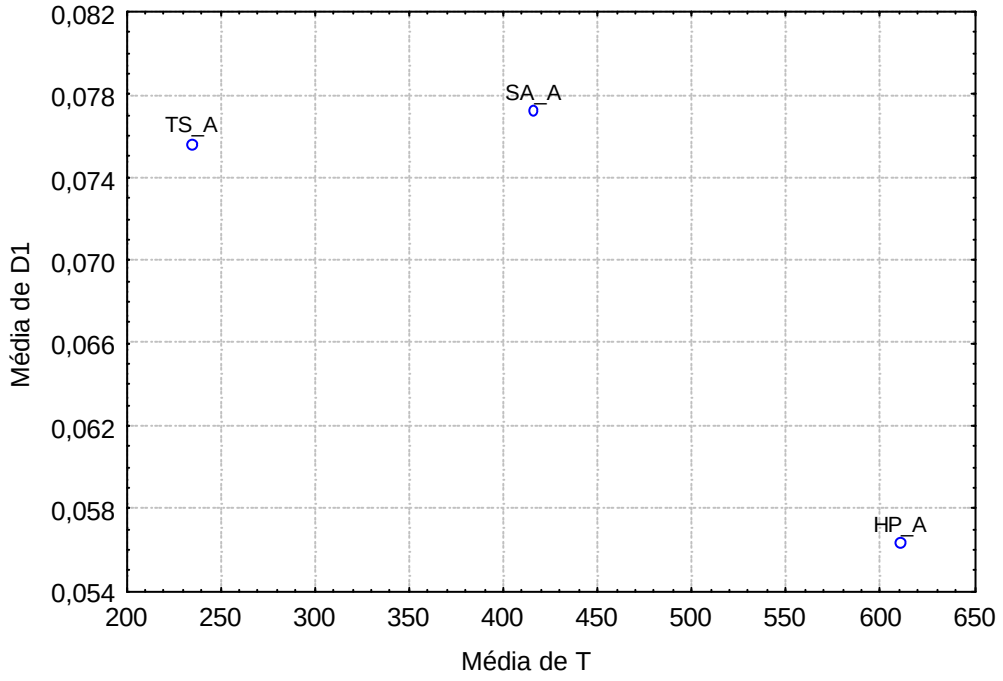


Figura 7.58 Hibridização e paralelização de alto nível (configurações base com melhor qualidade de aproximação média) - Diagrama de qualidade de aproximação e tempo de execução por algoritmo

Algoritmo	D1			T		
	Média	Desvio padrão	Nº de instâncias em que é signif. melhor	Média (s)	Desvio padrão (s)	Nº de instâncias em que é signif. menor
SA_A	0.077	0.036	7	415.82	362.32	6
TS_A	0.076	0.031	8	234.72	152.15	**9**
HP_A	0.056	0.032	**9**	610.62	433.69	4

Tabela 7.37 Hibridização e paralelização de alto nível (configurações base com melhor qualidade de aproximação média) - Desempenho global

7.9.2 Configurações base com pior qualidade de aproximação média

O par de algoritmos com pior qualidade de aproximação média, que será usado na configuração híbrida e paralela de alto nível HP_B, é constituído por: TS_B, com (8, 15, 5), e SA_B, com (8, 2, 200) e subvizinhança 2.

As tabelas ANOVA para os valores de D1 e do tempo de execução obtidos com as 3 configurações, são apresentadas, respectivamente, na Tabela 7.35 e na Tabela 7.36.

Fonte	GL Fonte	DQM Fonte	GL Erro	DQM Erro	F	Valor de prova (%)
Instância	8	0.0425	243	0.0010	43.9630	**0.00**
Algoritmo	2	0.0057	16	0.0041	1.4090	27.31
Instância * Algoritmo	16	0.0041	243	0.0010	4.2026	**0.00**

Tabela 7.38 Hibridização e paralelização de alto nível (configurações base com pior qualidade de aproximação média) - Tabela ANOVA para D1

Fonte	GL Fonte	DQM Fonte	GL Erro	DQM Erro	F	Valor de prova (%)
Instância	8	3.20E+05	243	1.06E+04	30.2855	**0.00**
Algoritmo	2	2.84E+05	16	1.71E+04	16.5583	**0.01**
Instância * Algoritmo	16	1.71E+04	243	1.06E+04	1.6186	6.46

Tabela 7.39 Hibridização e paralelização de alto nível (configurações base com pior qualidade de aproximação média) - Tabela ANOVA para o tempo de execução

Para a distância D1, o factor *Algoritmo*, individualmente, não é significativo. Para o tempo de execução, não há uma interacção significativa entre *Instância* e *Algoritmo*.

No diagrama de médias da Figura 7.59 e o diagrama do tipo caixa da Figura 7.61, confirma-se graficamente a inexistência de diferenças significativas na qualidade das aproximações obtidas com as 3 configurações. Na análise *post hoc*, com testes de Tukey, apenas

o algoritmo SA_B está ausente (em duas instâncias) do grupo de algoritmos de maior qualidade, e sem diferenças significativas entre eles em todas as instâncias.

A configuração HP_B apresenta um tempo de execução significativamente superior às configurações base (Figura 7.60 e Figura 7.62), evidenciando-se um ligeiro aumento da dispersão com o tempo de execução médio. Uma análise *post hoc*, com testes de Tukey, revela que há diferenças significativas (no sentido de um aumento do tempo de execução) entre esta configuração e as restantes em duas instâncias.

Com a hibridização das configurações base com pior qualidade de aproximação média não se registou alteração da qualidade das aproximações, enquanto ao nível do tempo de execução se verificou um aumento.

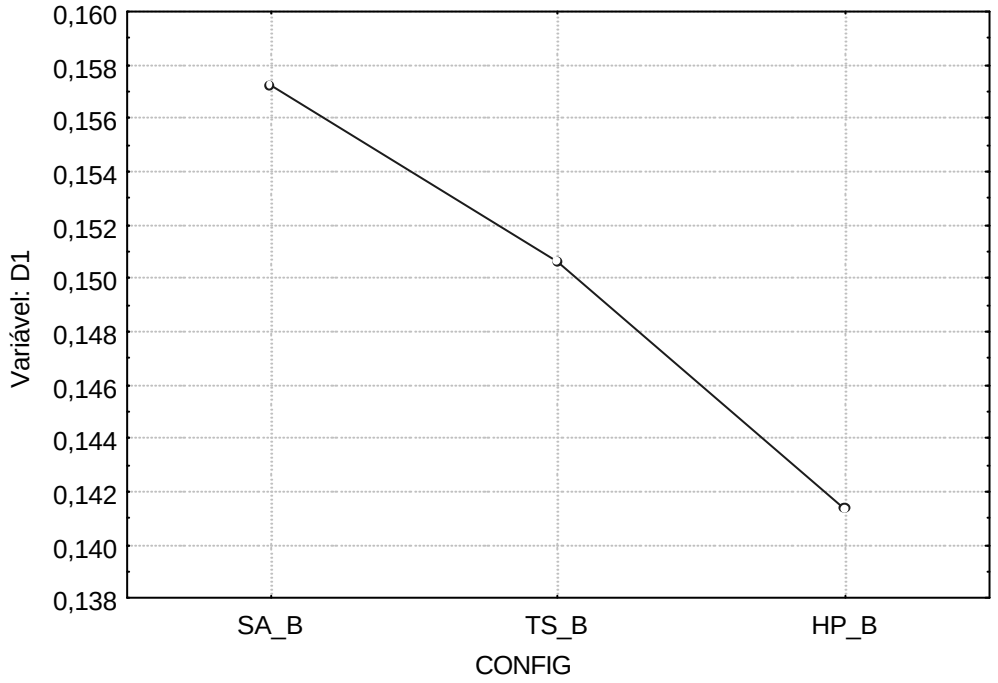


Figura 7.59 Hibridização e paralelização de alto nível (configurações base com pior qualidade de aproximação média) - Diagrama de médias de D1 para *Algoritmo*

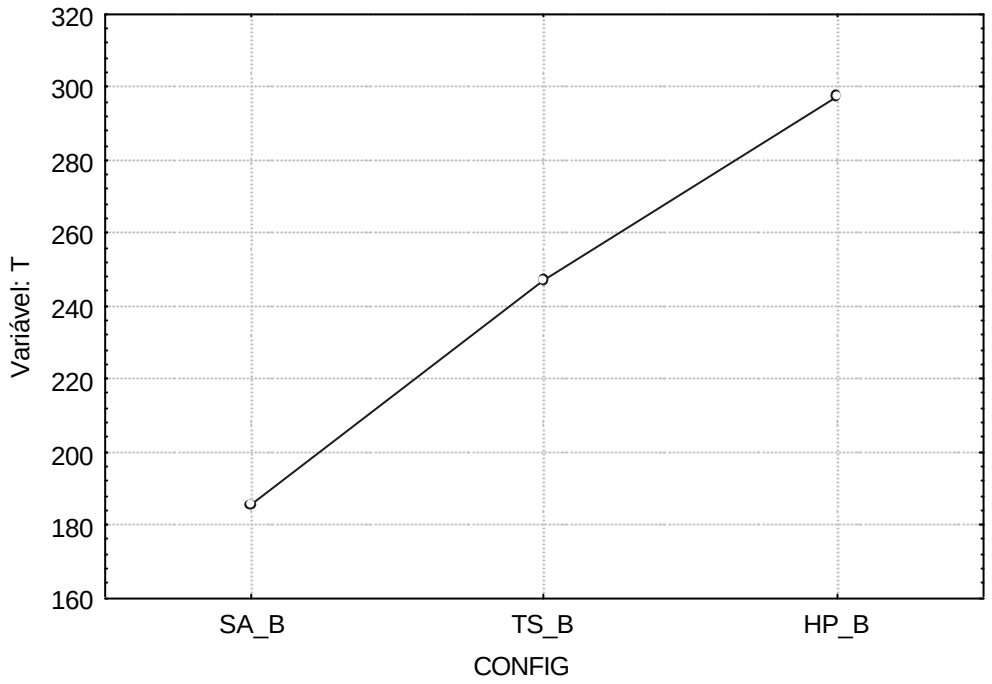


Figura 7.60 Hibridização e paralelização de alto nível (configurações base com pior qualidade de aproximação média) - Diagrama de médias do tempo de execução para *Algoritmo*

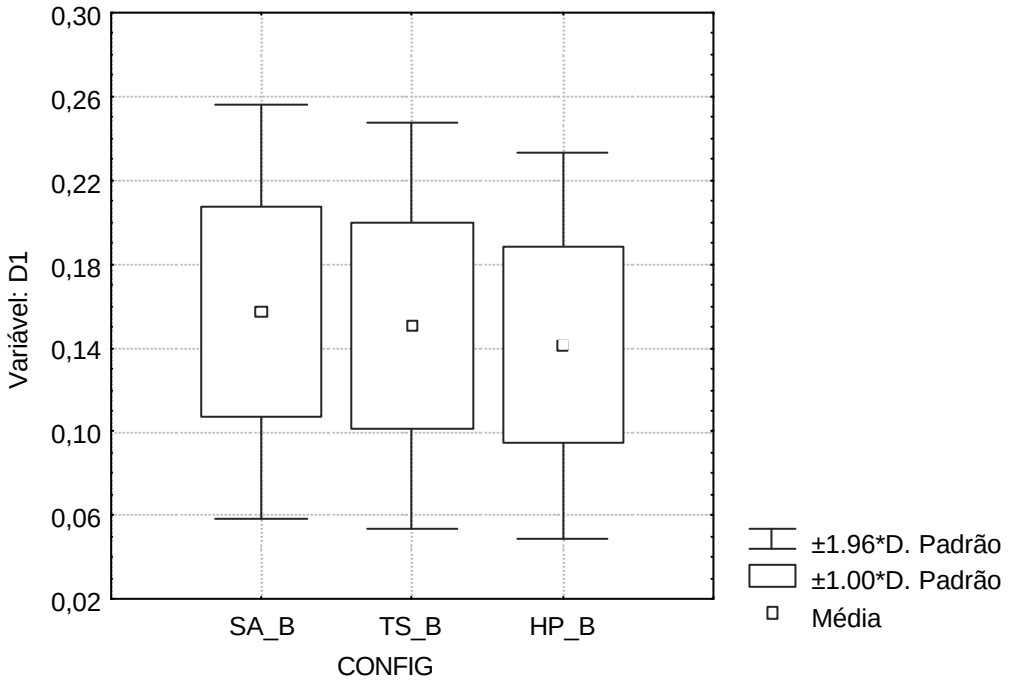


Figura 7.61 Hibridização e paralelização de alto nível (configurações base com pior qualidade de aproximação média) - Diagrama do tipo caixa de D1 para *Algoritmo*

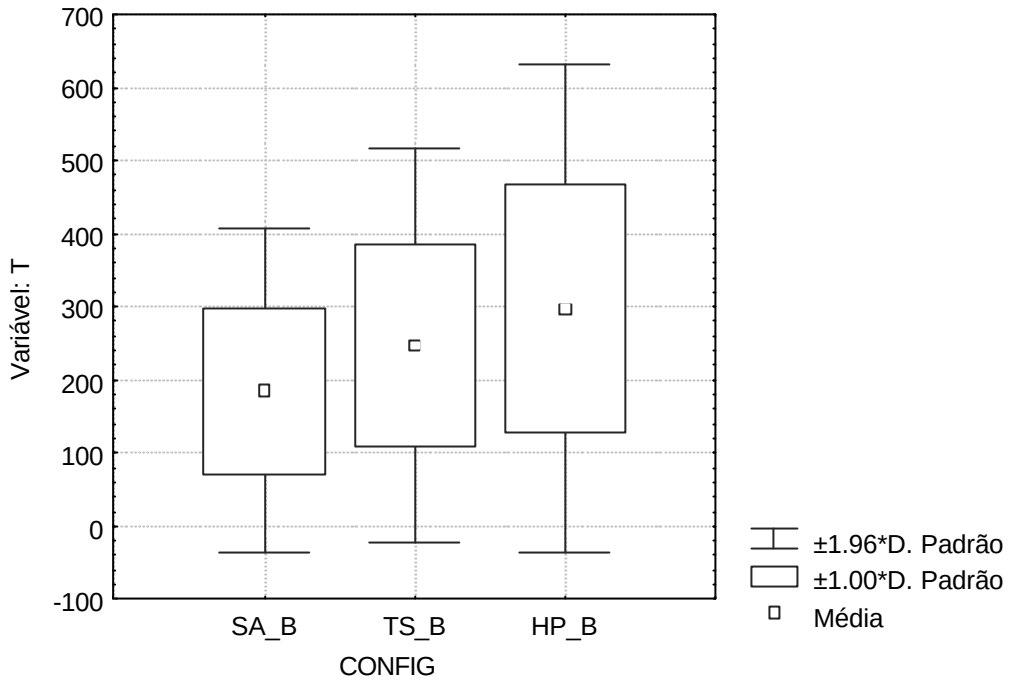


Figura 7.62 Hibridização e paralelização de alto nível (configurações base com pior qualidade de aproximação média) - Diagrama do tipo caixa do tempo de execução para *Algoritmo*

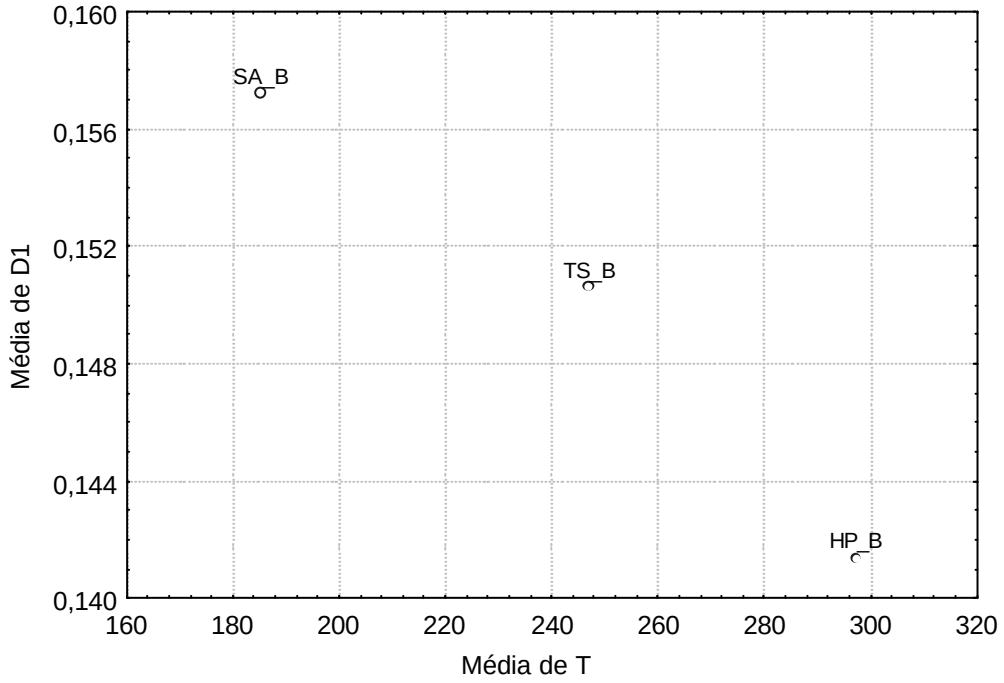


Figura 7.63 Hibridização e paralelização de alto nível (configurações base com pior qualidade de aproximação média) - Diagrama de qualidade de aproximação e tempo de execução por algoritmo

Algoritmo	D1			T		
	Média	Desvio padrão	Nº de instâncias em que é signif. melhor	Média (s)	Desvio padrão (s)	Nº de instâncias em que é signif. menor
SA_B	0.157	0.050	7	185.36	113.41	**9**
TS_B	0.151	0.049	**9**	247.07	137.52	**9**
HP_B	0.141	0.047	**9**	297.43	170.27	7

Tabela 7.40 Hibridização e paralelização de alto nível (configurações base com pior qualidade de aproximação média) - Desempenho global

7.10 Síntese dos resultados computacionais

A apreciação global apresentada nesta secção e as análises conduzidas individualmente nas secções anteriores, incidem sobre um conjunto de experiências em que se realizam variações sobre um conjunto de factores, sendo fixado um outro conjunto de factores, de acordo com critérios anteriormente expostos. É neste contexto, que envolve factores fixos como os tipos de problema e instâncias, o critério de paragem, ou o próprio processo de geração de soluções iniciais, que se situam as conclusões aqui apresentadas.

Versões básicas de MOTS* e PSA

Para as configurações da versão básica de MOTS* analisadas, verificou-se que os valores médios da qualidade de aproximação se ordenam aproximadamente da mesma forma nas várias instâncias, variando sobretudo a proximidade relativa desses valores. Um efeito idêntico é observável para o tempo de execução. Também na versão básica de PSA se observa este efeito, para o tempo de execução. No entanto, no caso da versão básica de PSA, em virtude de não existirem diferenças significativas de qualidade de aproximação, entre a grande maioria das configurações, o mesmo efeito não é verificável.

O aumento do número de soluções da população de 8 para 16 conduz a melhorias na qualidade de aproximação e a uma redução na dispersão dos valores obtidos. Para ambas as versões básicas, praticamente não há diferenças significativas entre as configurações com populações de 16 soluções, enquanto nas configurações com populações de 8 soluções, as diferenças aparecem de forma mais assinalável. Estes efeitos são bastante mais significativos na versão básica de MOTS* do que na versão básica de PSA. Em nenhum dos casos é estatisticamente significativo o efeito esperado de aproximação dos valores médios das distâncias D1.

Ao nível do tempo de execução, os valores médios são maiores para as configurações com populações de 16 soluções, incremento este que é acompanhado pela dispersão dos valores obtidos. Relativamente às diferenças entre configurações com o mesmo número de soluções, o comportamento é simétrico do comportamento para a qualidade de

aproximação: praticamente não há diferenças significativas entre as várias configurações com populações de 8 soluções, enquanto nas configurações com populações de 16 soluções, essas diferenças se tornam mais significativas.

Verifica-se uma forte interacção entre os factores variáveis da versão básica de MOTS*: comprimento da lista e dimensão da subvizinhança. É estatisticamente significativa a redução da qualidade de aproximação para configurações com subvizinhanças de dimensão reduzida e listas de dimensão elevada, e ainda para configurações sem lista e com subvizinhanças elevadas. O tempo de execução aumenta com a dimensão da lista e a dimensão da subvizinhança.

Para os factores variáveis da versão básica de PSA, temperatura inicial e comprimento do *plateau*, verifica-se igualmente a existência de um interacção significativa. É estatisticamente significativa a redução da qualidade de aproximação em configurações com temperaturas iniciais muito elevadas e arrefecimento lento. O tempo de execução aumenta com a temperatura inicial e diminui com a velocidade de arrefecimento, indicando que as configurações com estas características têm um maior tempo de convergência.

Para ambas as versões básicas, as configurações com menores valores médios da qualidade de aproximação exibem também as menores dispersões, sucedendo o inverso com o tempo de execução. Verifica-se ainda que este cresce com o número de soluções não-dominadas de cada instância.

Analisando conjuntamente as medidas de desempenho consideradas, constata-se que não existem diferenças estatisticamente significativas entre muitas das configurações consideradas, para cada uma das versões básicas consideradas, sobretudo para o PSA. No caso particular da versão básica de MOTS*, a uma melhor qualidade de aproximação está associado um maior tempo de execução, enquanto para a versão básica de PSA esta associação não é estatisticamente significativa. Um futuro estudo da versão básica de PSA, com um critério de paragem que permita convergências mais lentas, será útil no sentido de verificar se se mantém o nível de indiferenciação patenteado no estudo realizado.

Integração de abordagens de flexibilização

Algoritmo	Configuração base	Qualidade de Aproximação			Tempo de Execução		
		Aumento	Manutenção	Diminuição	Diminuição	Manutenção	Aumento
PSA com subvizinhança	pior	´					´
	melhor		´				´
PSA com lista de candidatos	pior	´				´	
	melhor		´				´
MOTS* com lista de candidatos	pior	´					´
	melhor			´	´		
PSA com vizinhança variável	pior		´			´	
	melhor		´				´
MOTS* com vizinhança variável	pior		´			´	
	melhor			´			´
Hibridização e paralelização de alto nível	pior		´				´
	melhor	´					´

Tabela 7.41 Abordagens de flexibilização - Quadro resumo dos resultados dos testes

A utilização de *subvizinhanças* em PSA permitiu, ao nível da qualidade de aproximação, colocar a configuração básica com pior média ao nível da melhor, com um aumento do tempo de execução para valores também da mesma ordem de grandeza. Estas configurações, assim obtidas, situam-se na mesma área de compromisso que as configurações da versão básica de MOTS* com menor qualidade e menor tempo de execução.

A integração da *lista de candidatos* teve efeitos diferentes sobre cada uma das versões básicas. No caso do PSA, esta integração não teve efeitos significativos sobre o desempenho da configuração com melhor qualidade de aproximação média. No entanto, permitiu melhorar a qualidade de aproximação da configuração com pior qualidade de aproximação média, colocando-a ao nível da melhor, com um aumento do tempo de execução também para a mesma ordem de valores. Este mesmo efeito tem lugar também com a integração com MOTS*. Contudo, a configuração com melhor qualidade de aproximação média é pior, em termos da qualidade de aproximação, acompanhada de uma redução do tempo de execução.

Para a *vizinhança variável* utilizada, não se registaram alterações significativas em nenhum dos aspectos, para as configurações baseadas em PSA. Relativamente às configurações baseadas em MOTS*, não se registam alterações significativas na configuração com melhor qualidade de aproximação média, enquanto na outra configuração se verifica uma degradação da qualidade de aproximação, sem alteração significativa do tempo de execução. O alargamento deste estudo a outros tipos de problemas, bem como a consideração de outros

tipos de vizinhanças variáveis será útil no sentido de obter resultados mais gerais sobre o interesse da integração deste tipo de estratégia.

As configurações de *hibridização e paralelização de alto nível* apresentam, no caso das configurações base de melhor qualidade de aproximação média, uma ligeira melhoria da qualidade de aproximação, acompanhada de um aumento do tempo de execução, em relação às configuração base de PSA. Não se verifica uma melhoria de qualidade em relação às configurações base de MOTS*. No caso das configurações base de pior qualidade de aproximação média, não se regista qualquer alteração significativa da qualidade de aproximação, verificando-se, no entanto, um aumento do tempo de execução.

O estudo preliminar realizado permite verificar, em geral, a existência de interacção entre as configurações dos algoritmos base e as abordagens de flexibilização, pelo que se justifica um aprofundamento da análise relativamente a esta interacção.

7.11 Conclusões

Procurou-se, neste capítulo, conduzir um estudo computacional rigoroso e exaustivo, para o que se consideraram as orientações de [Barr et al. 1995], para experiências computacionais na área dos métodos heurísticos. A utilização de um conjunto de métodos estatísticos, em particular a análise de variância, desempenhou um papel fundamental na análise de resultados, tendo permitido estudar com rigor a influência de diversos factores variáveis no comportamento das configurações algorítmicas consideradas, ao nível da qualidade de aproximação e do tempo de execução.

Na secção anterior resumiram-se os principais resultados dos testes computacionais, destacando-se aqui apenas as conclusões mais gerais.

Para a versão básica de MOTS*, a influência dos diversos factores variáveis é bastante clara e significativa, enquanto para a versão básica de PSA não existem diferenças significativas ao nível da qualidade de aproximação entre a maioria das configurações; os resultados obtidos com a integração da *lista de candidatos* e com a *hibridização e paralelização de alto nível* são encorajadores, enquanto com a integração da *vizinhança variável* não se registaram quaisquer melhorias de desempenho. A interacção forte que se verificou existir entre as configurações do algoritmo base e as abordagens de flexibilização será talvez um dos assuntos mais pertinentes para aprofundamento em estudos futuros.

Os testes às abordagens de flexibilização contribuíram igualmente para uma primeira validação do potencial de aplicação do *framework* como instrumento para a comparação de algoritmos. A utilização do paradigma OO permitiu que, de forma natural e com um esforço reduzido, as abordagens de flexibilização fossem implementadas como extensões muito

localizadas ao *template* de pesquisa local multiobjectivo, reutilizando muitas das classes já aí presentes, ou como extensões de alto nível envolvendo a execução híbrida e paralela dos algoritmos básicos. Sendo a grande maioria do código comum às várias abordagens, a influência de particularidades de codificação é praticamente anulada e o isolamento dos efeitos que se pretendem testar é melhor conseguido.

Os variados testes computacionais realizados demonstraram que o *framework* permite uma fácil especificação da configuração algorítmica. Esta facilidade poderá naturalmente ser de grande interesse prático na "construção" de meta-heurísticas de qualidade para problemas práticos nos quais, talvez mais do que no presente estudo, a configuração e as possibilidades de hibridização desempenhem um papel mais importante.

8
CONCLUSÕES E DESENVOLVIMENTOS FUTUROS

Como conclusão para esta dissertação, apresenta-se, na secção 8.1 uma descrição sucinta do trabalho realizado, seguida de uma enumeração dos principais resultados obtidos, na secção 8.2, uma discussão sobre o nível de satisfação dos objectivos estabelecidos para o trabalho, na secção 8.3, e algumas conclusões de carácter geral, na secção 8.4. Os possíveis desenvolvimentos futuros finalizam o capítulo, na secção 8.5.

8.1 Trabalho realizado

Nos capítulos iniciais da dissertação (2 e 3) apresenta-se uma sistematização do domínio das meta-heurísticas e, em particular, das meta-heurísticas multiobjectivo, que permitiu:

- estabelecer uma parte significativa do contexto no qual este trabalho se inscreve;
- concretizar uma análise do domínio, fundamental para o processo de desenvolvimento do framework;
- identificar um conjunto importante de linhas de evolução na concepção de meta-heurísticas, que se designou por flexibilização em meta-heurísticas;
- propor estratégias genéricas de flexibilização, utilizando vizinhanças variáveis e listas de candidatos, para meta-heurísticas baseadas em pesquisa local, e a sua extensão a meta-heurísticas baseadas em pesquisa local multiobjectivo;
- desenvolver um template em que se identificam os aspectos comuns das meta-heurísticas baseadas neste princípio;
- propor, igualmente para este tipo de meta-heurísticas, uma abordagem de hibridização e paralelização de alto nível.

A necessidade de aproximar a teoria e a aplicação e de criar formas mais eficientes de implementação e comparação de métodos na área das meta-heurísticas, tem motivado o aparecimento de diversas abordagens orientadas por objectos para meta-heurísticas, caracterizadas no capítulo 4. No capítulo 5 propôs-se um *framework* orientado por objectos para meta-heurísticas multiobjectivo que, no seu estado actual de desenvolvimento, foca, em particular, a área da pesquisa local multiobjectivo. Também este trabalho foi realizado na sequência das motivações apontadas. Em particular, justifica-se pela inexistência de qualquer proposta de abordagem para meta-heurísticas multiobjectivo e pelo facto de na literatura não ser dado ênfase suficiente ao facto de que o paradigma OO é particularmente adequado para a implementação e integração, com pesquisa local, de estratégias genéricas de flexibilização como listas de candidatos, vizinhanças variáveis ou hibridização e paralelização de alto nível.

A aplicação a um problema de escalonamento multiobjectivo de tarefas numa máquina, descrita no capítulo 6, permitiu completar a descrição do *framework*, no que se refere à sua utilização. A realização de um conjunto de testes computacionais, cuja concepção e resultados são apresentados no capítulo 7, permitiu, entre outros objectivos, realizar uma primeira avaliação do potencial interesse da utilização das estratégias genéricas de flexibilização propostas.

8.2 Resultados obtidos

Os principais resultados alcançados com o trabalho descrito nesta dissertação são os seguintes:

- Definição de uma abordagem para concepção e implementação de meta-heurísticas multiobjectivo, integrando estratégias de flexibilização genéricas, com base num *framework* orientado por objectos. Esta abordagem diferencia-se de outras abordagens por propor a utilização explícita do paradigma OO para suportar flexibilização em meta-heurísticas e constituir a primeira abordagem orientada por objectos para meta-heurísticas multiobjectivo.
- Generalização de estratégias de lista de candidatos elite e de vizinhanças variáveis e respectiva integração com meta-heurísticas baseadas em pesquisa local.
- Definição de um *template* de pesquisa local multiobjectivo.
- Adaptação das estratégias genéricas definidas a contextos multiobjectivo e respectiva integração com meta-heurísticas baseadas em pesquisa local multiobjectivo.
- Definição de uma abordagem de hibridização e paralelização de alto nível baseada em pesquisa local multiobjectivo.
- Concepção de um *framework* e sua documentação, de acordo com as orientações gerais propostas por [Johnson 1992], incluindo: o objectivo, ou seja, o domínio para o qual foi concebido; o desenho detalhado, com estruturas de classes e respectivas colaborações, e a utilização de *padrões de desenho*; instruções detalhadas sobre a utilização do *framework* para construir aplicações, com a aplicação a um problema de escalonamento multiobjectivo de tarefas numa máquina.
- Implementação do *framework* em C++.

A abordagem foi aplicada a um problema de escalonamento multiobjectivo de tarefas numa máquina, e na realização de testes computacionais sobre este mesmo problema, o que permitiu verificar o potencial do *framework* nas sua várias aplicações possíveis.

8.3 Grau de satisfação dos objectivos estabelecidos

Os principais objectivos estabelecidos para o trabalho descrito nesta dissertação foram:

- Propor estratégias genéricas de flexibilização em meta-heurísticas multiobjectivo, e demonstrar o interesse da sua utilização.
- Desenvolver um *framework* orientado por objectos que disponibilize uma arquitectura de base flexível para a construção, aplicação e comparação de meta-heurísticas multiobjectivo, integrando estratégias de flexibilização genéricas.

O primeiro objectivo foi fundamentalmente satisfeito da seguinte forma:

- A sistematização do domínio realizada nos capítulos 2 e 3 permitiu identificar e caracterizar, por um lado, os princípios meta-heurísticos e principais meta-heurísticas "básicas", e, por outro, um conjunto de tendências de flexibilização. De uma análise criativa do enquadramento resultante, em interacção com o próprio desenvolvimento do *framework*, surgiram as soluções de integração das estratégias genéricas de flexibilização consideradas e de adaptação a contextos multiobjectivo.
- A aplicação ao problema considerado como caso de estudo, e a execução de testes computacionais exaustivos permitiu realizar uma validação preliminar do interesse da utilização das estratégias consideradas. Enquanto que, para a utilização de listas de candidatos e hibridização e paralelização de alto nível, os resultados são desde já encorajadores, para as vizinhanças variáveis verificou-se ser necessário aprofundar a análise, para poder elaborar uma apreciação mais definitiva.

Relativamente ao segundo objectivo, as contribuições fundamentais para a sua satisfação foram as seguintes:

- O elevado grau de flexibilidade e configurabilidade, disponibilizado pela utilização do paradigma OO, em particular pela aplicação de *padrões de desenho*, pôde ser constatado de diversas formas, desde a definição, sobre o *template* de pesquisa local multiobjectivo, das abordagens "básicas" baseadas neste princípio, às extensões relativas às estratégias genéricas de flexibilização, realizadas de forma simples e com um elevado nível de reutilização, ou à multiplicidade de configurações algorítmicas possíveis de obter.
- A clareza conceptual da infra-estrutura, ligada à modelação efectiva do domínio e à separação de facetas, esteve, em parte, na origem das soluções de integração das

estratégias genéricas de flexibilização consideradas, bem como na origem da sua adaptação a um contexto multiobjectivo.

- A aplicação ao problema considerado, num cenário de experimentação, com o nível de reutilização permitido, exigiu um esforço relativamente reduzido, na ordem de 1 pessoa-mês, com o desenvolvimento de cerca de 20 classes, correspondentes a aproximadamente 35% do número total de linhas de código (do conjunto do *framework* e da aplicação).
- As abordagens de flexibilização foram implementadas como extensões muito localizadas ao *template* de pesquisa local multiobjectivo, reutilizando muitas das classes já aí presentes, ou como extensões de alto nível envolvendo a execução híbrida e paralela dos algoritmos básicos. Por outro lado, os aspectos particulares ao problema são definidos num mesmo conjunto de classes. Com a partilha de componentes e métodos, a influência de particularidades de codificação é praticamente anulada e o isolamento dos efeitos que se pretendem testar é melhor conseguido. Constitui-se, assim, uma plataforma mais adequada para comparação de meta-heurísticas, comparação esta que foi realizada para a validação preliminar do interesse da utilização das estratégias genéricas de flexibilização consideradas.

8.4 Conclusões gerais

Neste trabalho apresenta-se um conjunto de contribuições inovadoras no sentido de algumas linhas de evolução mais recentes na concepção de meta-heurísticas, que se baseiam na introdução de mecanismos de modificação dos seus componentes e estratégias elementares, e que se designaram por *flexibilização em meta-heurísticas*. A utilidade desta perspectiva, do ponto de vista da concepção e implementação de meta-heurísticas, foi confirmada, sobretudo pelo facto de ter orientado a estruturação de conceitos adoptada no desenho de um *framework*, com o qual se pretende disponibilizar uma ferramenta de base genérica e flexível, para a construção, aplicação e comparação de meta-heurísticas multiobjectivo

O *framework* foi aplicado a um problema de escalonamento multiobjectivo de tarefas numa máquina, para minimização do atraso máximo, do número de tarefas atrasadas e da soma ponderada dos atrasos. A aplicação foi realizada com um esforço relativamente reduzido e permitiu concluir da adequação do *framework* na sua vertente de modelação dos componentes dependentes do problema.

Conforme referido na secção anterior, foi possível, com o trabalho desenvolvido, concluir do interesse da utilização de estratégias genéricas de flexibilização em meta-heurísticas multiobjectivo. Os resultados obtidos nos testes computacionais, com a integração da *lista de candidatos* e com a *hibridização e paralelização de alto nível*, são encorajadores. Com a integração da

vizinhança variável não foi possível registar melhorias de desempenho, pelo que será de todo o interesse continuar o estudo nesta área mais específica. Os testes conduzidos com as versões básicas, além de contribuírem para a avaliação do impacto das estratégias de flexibilização, permitiram igualmente uma caracterização do comportamento que, no caso da MOTS*, se verificou ser claramente dependente dos factores variáveis, enquanto no caso do PSA se verificou ser, para a qualidade das aproximações, em geral, indiferenciado.

Finalmente, e tal como foi salientado na secção anterior, o trabalho desenvolvido permitiu concluir da adequação do paradigma OO para disponibilizar as características pretendidas para a abordagem e, em particular, para a implementação e integração, com pesquisa local, de estratégias genéricas de flexibilização como listas de candidatos, vizinhanças variáveis ou hibridização e paralelização de alto nível.

8.5 Desenvolvimentos futuros

Ao longo do trabalho realizado foram surgindo várias pistas para desenvolvimentos futuros, de entre as quais se destacam aqui as seguintes:

- Realização de uma análise do *framework* do ponto de vista da eficiência, com o objectivo de melhorar o compromisso generalidade - eficiência. Esta melhoria poderá passar por um refinamento do código em algumas áreas do *framework*, e pelo redesenho no sentido de, nas áreas mais críticas, privilegiar o polimorfismo estático em alternativa aos mecanismos de herança.

- Integração de outras abordagens de pesquisa local e de abordagens baseadas em recombinação. A extensão do *framework* para disponibilizar abordagens baseadas em *GRASP* ou *VNS* será relativamente simples, uma vez que os respectivos componentes básicos estão já largamente desenvolvidos no *framework*. A inclusão de abordagens baseadas em recombinação beneficiará igualmente de alguns conceitos já integrados no *framework*, em particular o conceito de população. Terá que ser objecto de avaliação aprofundada a opção entre considerar um novo *template* para recombinação multiobjectivo, baseado numa generalização do *template* da *Scatter Search*, ou generalizar o *template* de pesquisa local multiobjectivo, de modo idêntico ao realizado para contextos de objectivo único em [Vaessens et al. 1995].

- Utilização do *framework* em Sistemas de Apoio à Decisão. Este é o objectivo último do desenvolvimento do *framework*. Além de um esforço do autor no sentido de procurar a concretização desta aplicação, procurar-se-á construir um projecto *Open Source* em torno do *framework*, disponibilizando-o para utilização em trabalhos de índole teórica ou prática e procurando obter, assim, informação

relevante para o seu aperfeiçoamento no sentido das várias utilizações de que poderá ser alvo.

- Integração de funcionalidade relacionada com a realização de testes com elevado grau de automatização, sofisticando, em particular, a recolha e disponibilização de informação para análise estatística, e criando uma camada para a especificação a alto nível de baterias de testes.
- Disponibilização de configurabilidade de alto nível das abordagens meta-heurísticas incluídas no *framework*, num interface próximo a uma linguagem de alto nível como a linguagem *Localizer* [Michel, van Hentenryck 1998], mas enquadrado pelo paradigma OO.
- Desenvolvimento do estudo das abordagens genéricas de flexibilização e dos aspectos deixados em aberto com o estudo computacional: o elevado grau de indiferenciação entre as configurações de PSA, no que respeita à qualidade das aproximações; a ausência de melhorias de qualidade das aproximações, nas configurações com *vizinhanças variáveis*; a forte interacção entre as configurações base e as estratégias genéricas de flexibilização.

A
NOTAÇÃO USADA EM DIAGRAMAS

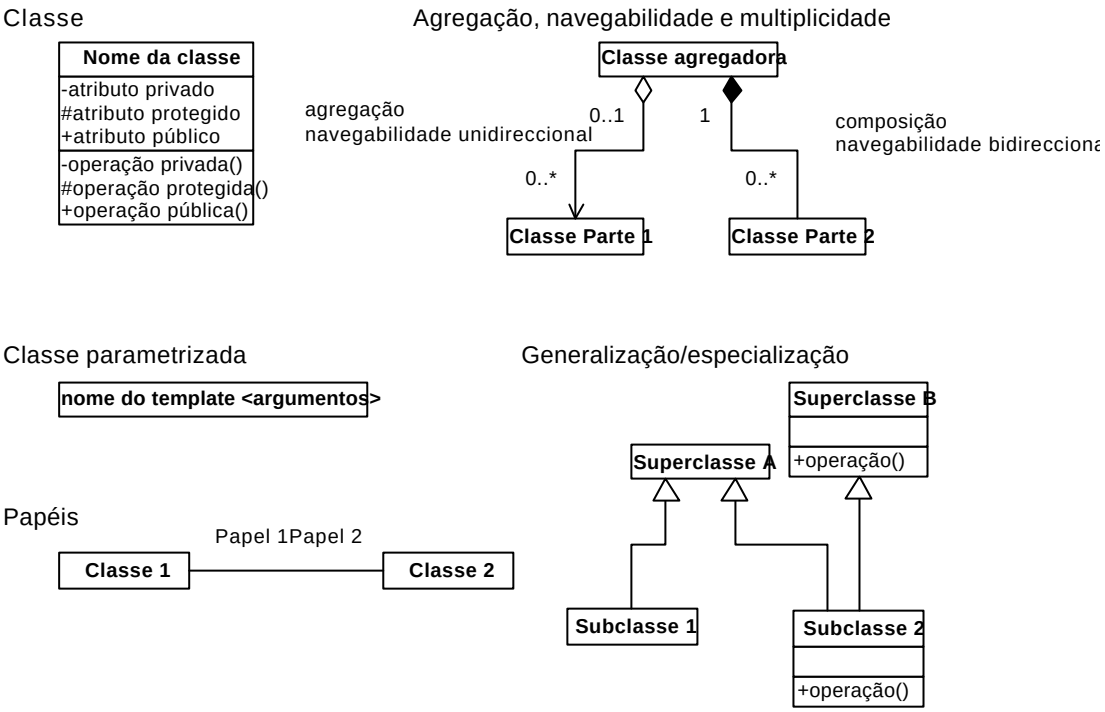


Figura A.1 Diagramas de classes

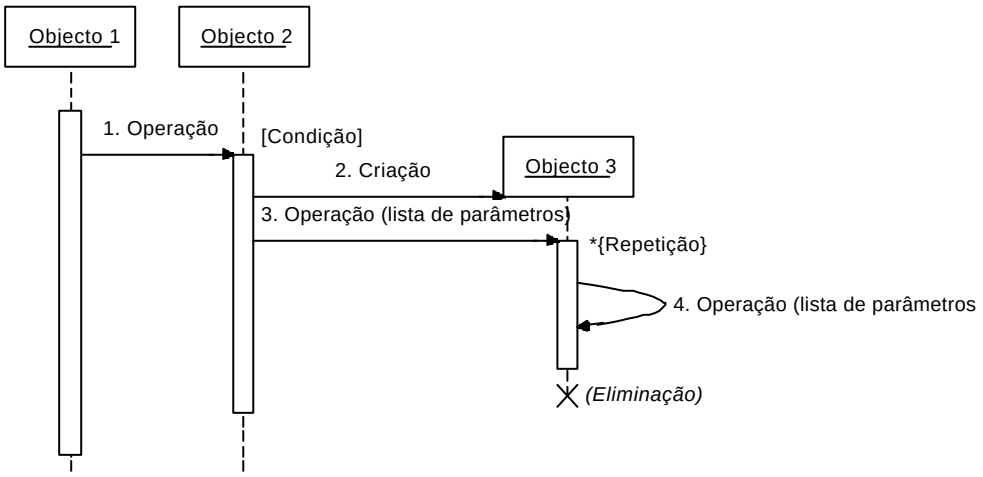


Figura A.2 Diagramas de sequência

B
COMPLEMENTO À DESCRIÇÃO DO *FRAMEWORK*

B.1 Introdução

Como referido no Capítulo 5, por motivos de espaço e organização do texto, nem todas as partes foram aí alvo de descrição, ao nível do desenho detalhado. Privilegiaram-se as partes directamente relacionadas com a pesquisa local multiobjectivo, localizando-se a descrição das restantes neste anexo.

Serão aqui focados, então, os seguintes aspectos:

1. bibliotecas de estruturas de dados e algoritmos utilizadas;
2. interface entre cliente e *framework*;
3. suporte para os dados do problema, baseado em modelação algébrica;
4. *solvers* meta-heurísticos;
5. algoritmos construtivos.

B.2 Bibliotecas de estruturas de dados e algoritmos

A reutilização de bibliotecas genéricas de estruturas de dados e algoritmos constituiu uma das linhas de orientação do desenvolvimento do *framework*, em consonância com os princípios de promoção da reutilização, nos quais se enquadra este trabalho.

Nesse sentido, foi realizado um levantamento que revelou a existência de duas bibliotecas com grande interesse, ambas disponibilizando um conjunto de componentes genéricos de C++ bem estruturados e integráveis com o *framework* de forma simples e transparente:

- A *Standard Template Library* (STL) [Musser, Saini 1996], da qual são utilizados contentores (vectores e listas), iteradores e alguns algoritmos de ordenação. A STL apresenta um compromisso interessante entre flexibilidade e eficiência, uma vez que todos os componentes genéricos têm uma implementação com um desempenho muito próximo das correspondentes rotinas codificadas de forma específica.

- A *Library of Efficient Data Types and Algorithms* (LEDA) [Mehlhorn, Näher 1995] disponibiliza um conjunto de tipos de dados e algoritmos, com preocupações de implementação eficiente para cada um dos tipos de dados. A biblioteca complementa a STL, em particular na disponibilização de componentes específicos para grafos. O *framework* recorre, em particular, às classes de geração de números aleatórios, de utilização simples e flexível.

B.3 Interface entre cliente e *framework*

A separação entre as características específicas dos problemas e os algoritmos meta-heurísticos é o elemento estruturante do interface entre cliente e *framework*. No METHOOD tal é realizado, à semelhança de [Andreatta et al. 1998], com recurso ao *padrão de desenho* Estratégia:

- são considerados separadamente o cliente, o algoritmo em si e os objectos que circulam entre ambos: dados do problema, componentes dependentes do problema, informação de configuração específica aos algoritmos e resultados da execução;
- é possível desenvolver uma hierarquia de *solvers*, dando expressão à possibilidade de um *solver* meta-heurístico ter diversas variantes, que serão, no entanto, conformes com um interface geral.

Estrutura e participantes

O diagrama de estrutura de classes apresentado na Figura B.1 ilustra a forma como este *padrão* é aplicado no METHOOD.

Os participantes nesta parte do *framework* são os seguintes:

- *Cliente* (*aClient*). Utiliza um *solver* meta-heurístico (*MetaHeuristicSolver*) para resolver um problema de Optimização Combinatória multi-objectivo. Passa ao *solver* parâmetros para configuração de cada um dos algoritmos que o *solver* irá executar (*AlgParameters*). Os resultados que recebe consistem numa aproximação ao conjunto de soluções eficientes do problema (*EfficientSet*).

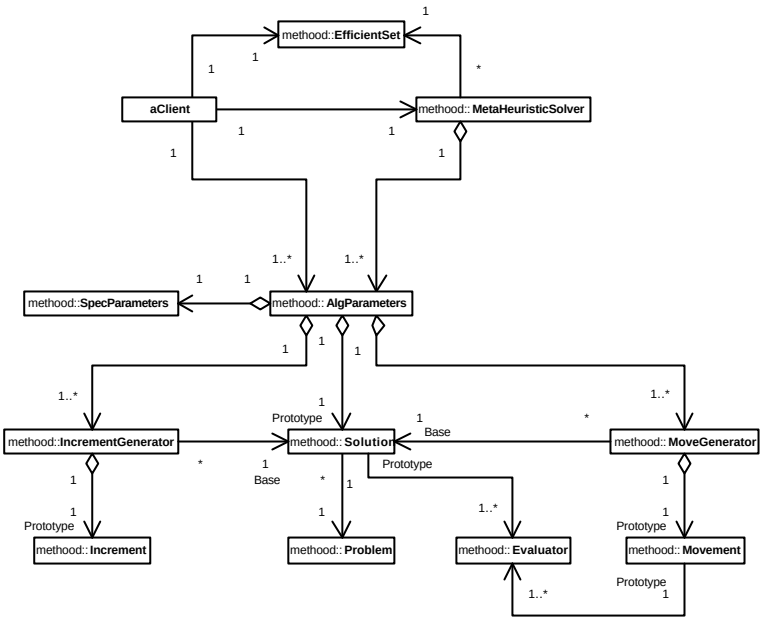


Figura B.1 Diagrama de classes para o interface entre cliente e *framework*

- *Parâmetros do algoritmo* (*AlgParameters*). Contém parâmetros específicos aos algoritmos (*SpecParameters*) e os componentes dependentes do problema: uma solução protótipo ($Solution_{Prototype}$), um conjunto de geradores de incrementos (*IncrementGenerator*) e um conjunto de geradores de movimentos (*MoveGenerator*).
- *Solução* (*Solution*). Contém a representação da solução, acedendo para tal à informação relativa aos dados do problema (*Problem*). Como protótipo ($Solution_{Prototype}$), permite a criação de todas as soluções de que o algoritmo necessitar. Estar-lhe-á associado um conjunto de funções de avaliação (*Evaluator*), uma para cada um dos objectivos considerados. A cada solução protótipo deverá estar associado um conjunto distinto de funções de avaliação, que ficará posteriormente associado a todas as soluções criadas a partir desta. Esta condição prende-se com a necessidade de evitar interacções entre componentes de diferentes algoritmos.
- *Dados do problema* (*Problem*). Centraliza o acesso aos dados do problema.
- *Gerador de incrementos* (*IncrementGenerator*). Cria incrementos de soluções, a partir de um protótipo de incremento ($Increment_{Prototype}$). É utilizado para a construção de soluções iniciais.
- *Incremento* (*Increment*). Contém a representação de um incremento à solução. Como protótipo ($Increment_{Prototype}$), permitirá a criação de novos incrementos a configurar pelo gerador de incrementos.

- *Gerador de movimentos* (*MoveGeneratorl*).
- *Movimento* (*Movement*). Também neste caso, a cada protótipo de movimento deverá estar associado um conjunto distinto de funções de avaliação, que ficará posteriormente associado a todos os movimentos criados a partir deste. Procura-se, assim, evitar interacções entre componentes de diferentes algoritmos.
- *Função de avaliação* (*Evaluator*).
- *Parâmetros específicos dos algoritmos* (*SpecParameters*). Contém informação específica não relacionada com o problema. Parte desta informação é comum a todos os algoritmos: critério de paragem, estratégia de vizinhança, estratégia de lista de candidatos. Outra parte conterá os elementos de configuração que diferem entre algoritmos, por exemplo, a configuração da lista tabu para a MOTS* e do esquema de arrefecimento para o PSA.
- *Aproximação ao conjunto de soluções eficientes* (*EfficientSet*). Num alargamento da descrição base, pode referir-se que conterá o resultado da execução do *solver* meta-heurístico, que consistirá das soluções eficientes do conjunto de soluções pesquisadas pelo *solver*.

Colaborações

A colaboração entre o cliente e o *framework* envolve a construção preliminar dos objectos que serão passados ao *solver* e a colaboração específica entre cliente e *solver*. A construção preliminar dos objectos será alvo de referência mais adiante, num ponto dedicado aos *solvers* meta-heurísticos. Apenas a interacção entre cliente e *solver* será aqui descrita. Esta interacção estrutura-se em duas fases (Figura B.2):

1. *Criação* (*CREATION*). O cliente cria um *solver* meta-heurístico concreto, com o qual interage e cuja execução é configurada através dos objectos que lhe são passados.
2. *Execução* (*EXECUTION*). Esta fase, que ocorre tipicamente logo após a criação do *solver*, lança a respectiva execução.

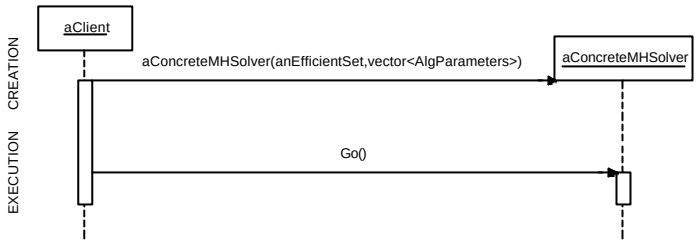


Figura B.2 Diagrama de sequência para a colaboração entre cliente e *solver*

B.4 Modelação algébrica dos dados do problema

A modelação algébrica é uma forma genérica de modelar um problema, que permite o seu tratamento por *solvers* genéricos. Neste tipo de modelação os dados do problema são representados por *parâmetros* que tomam *valores* de acordo com um conjunto de *índices* que os relacionam com objectos ou conjuntos de objectos particulares.

Por exemplo, num problema do caixeiro viajante, o *parâmetro* distância entre duas cidades d_{ij} será *indexado* à cidade de partida *i* e à cidade de destino *j*, tomando o *valor* adequado para cada um dos pares *i, j*.

Estrutura e participantes

Esta relação simples estrutura o conjunto de classes (Figura B.3) que permite no METHOOD a representação dos dados do problema.

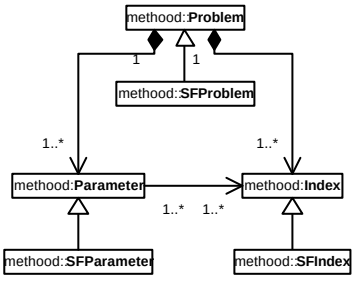


Figura B.3 Diagrama de classes para a representação dos dados do problema

Os participantes nesta parte do *framework* são os seguintes:

- *Problema* (*Problem*). Agrega as classes que realizam a representação dos dados do problema: os índices (*Index*) e os parâmetros (*Parameter*). Centraliza o acesso aos dados do problema
- *Índice* (*Index*). Contém informação por tipo de objecto do problema: uma identificação e os valores mínimo e máximo que o índice pode tomar.

- *Parâmetro* (*Parameter*). Contém valores de características de objectos (quando associado a um só índice) ou conjuntos de objectos (quando associado a vários índices). A um parâmetro estarão, consequentemente, associados um ou mais índices.

Esta estrutura permite um razoável grau de flexibilidade e de independência relativamente à origem dos dados, à sua representação ou à forma como se realiza o seu acesso, que são aqui considerados de forma abstracta. É assim possível comutar o acesso aos dados entre, por exemplo, ficheiros sequenciais ou bases de dados. Esta facilidade de comutação é suportada pela utilização de herança. Assim, seria possível, por exemplo, definir uma hierarquia de classes para acesso a informação de ficheiros sequenciais, e uma outra para bases de dados.

O METHOOD disponibiliza à partida uma hierarquia para acesso a ficheiros sequenciais (*SFProblem*, *SFIndex*, *SFParameter*). Estes ficheiros deverão respeitar o seguinte formato:

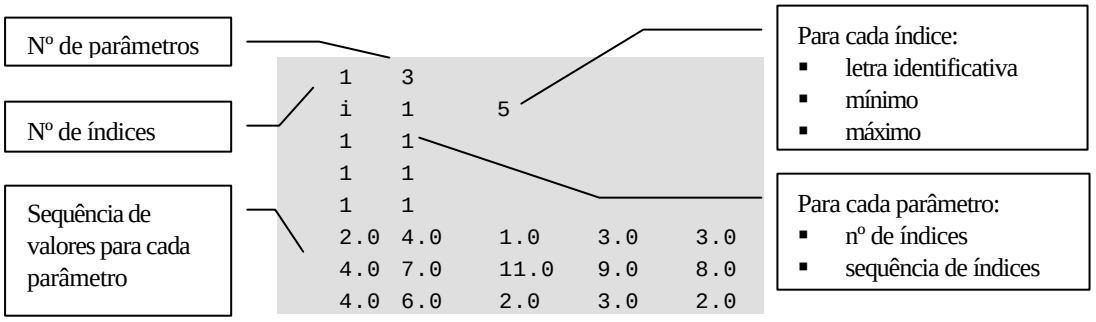


Figura B.4 Formato para o ficheiro sequencial de dados

O ficheiro exemplo apresentado na Figura B.4 teria a seguinte interpretação:

- O problema é descrito por 1 índice e 3 parâmetros.
- O índice 1 será identificado pela letra 'i' e assumirá valores entre 1 e 5.
- O parâmetro 1 é indexado pelo índice 1 (tendo, portanto, 5 valores).
- O parâmetro 2 é indexado pelo índice 1 (tendo, portanto, 5 valores).
- O parâmetro 3 é indexado pelo índice 1 (tendo, portanto, 5 valores).
- O parâmetro 1 assumirá os valores {2.0, 4.0, 1.0, 3.0, 3.0}.
- O parâmetro 2 assumirá os valores {4.0, 7.0, 11.0, 9.0, 8.0}.
- O parâmetro 3 assumirá os valores {4.0, 6.0, 2.0, 3.0, 2.0}.

Colaborações

A utilização da infra-estrutura de representação dos dados do problema é actualmente realizada em duas situações típicas:

1. *Carregamento de dados* (*LOADING*) (Figura B.5). Um cliente cria um problema, passando-lhe como parâmetro um controlador de leitura (da classe *ifstream*) para um ficheiro sequencial com os respectivos dados. A informação de configuração geral, que constitui o início do ficheiro, consiste no número de índices e de parâmetros. Para cada índice será lida, então, a respectiva informação. O mesmo sucederá, de seguida, para cada parâmetro. Entrar-se-á, então, num ciclo final de leitura dos valores relativos a cada parâmetro. Os valores são, neste caso particular, armazenados em vectores.

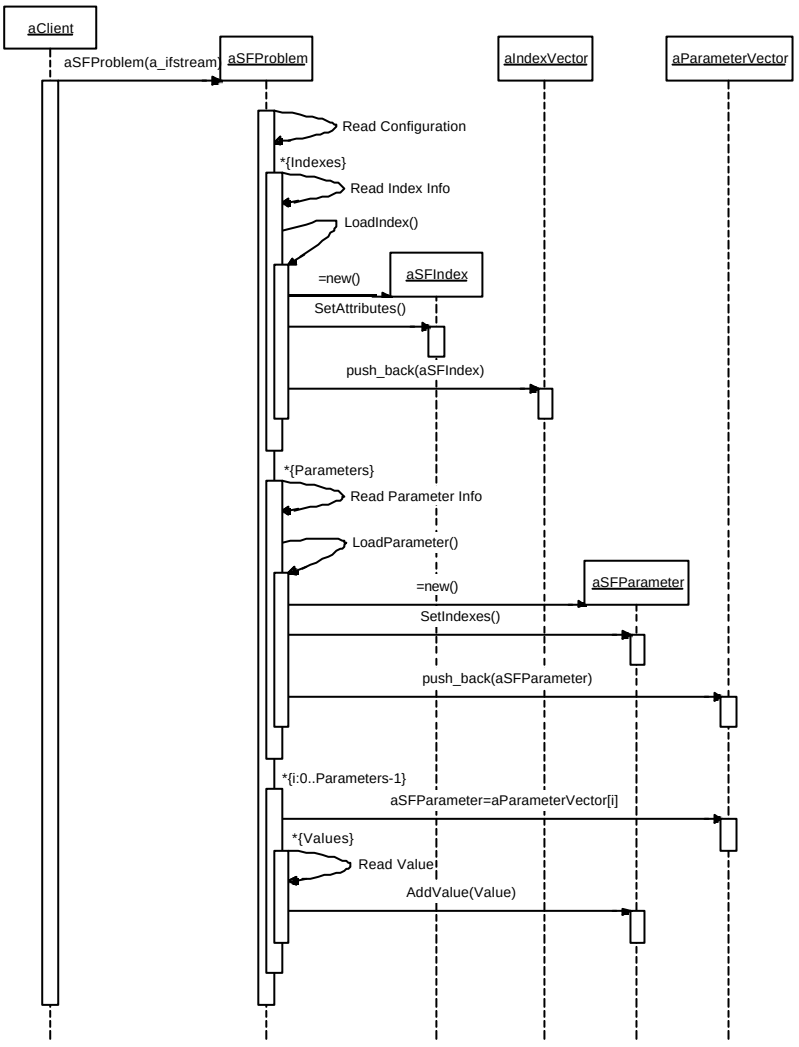


Figura B.5 Diagrama de sequência para o carregamento de dados do problema

2. *Leitura de dados* (*READING*) (Figura B.6). A leitura de dados realiza-se em duas etapas: em primeiro lugar, o cliente solicita, ao problema, o parâmetro desejado (identificado pela sua sequência no ficheiro de dados); de seguida, solicita, ao parâmetro, o valor particular desejado, identificado por um vector com os correspondentes valores particulares dos respectivos índices.

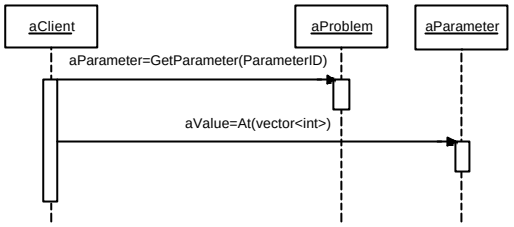


Figura B.6 Diagrama de sequência para a leitura de dados do problema

A consideração desta funcionalidade no METHOOD cobre uma lacuna existente nas restantes abordagens meta-heurísticas OO de utilização geral, que não apresentam suporte nesta área. Ao consagrar uma estrutura e um interface genérico para a definição dos dados, permite-se a aceleração e a simplificação de implementações. Esta formato pode, contudo, ser ultrapassado pelo cliente do *framework*, já que é garantida total flexibilidade para definir e utilizar outras formas de representação dos dados do problema.

Por outro lado, o *framework* aparece já dotado de um interface de leitura de ficheiros de texto sequenciais, um meio que é de utilização universal e frequente para importação e exportação de dados, com o qual se pretende contribuir igualmente para facilitar as tarefas de implementação.

B.5 *Solvers* meta-heurísticos

Conforme referido anteriormente, o *framework* disponibiliza, para além de algoritmos de pesquisa local multiobjectivo básicos, vizinhanças variáveis, estratégias de listas de candidatos e uma abordagem de paralelização e hibridização.

Estes mecanismos foram essencialmente implementados como extensões a uma parte mais nuclear do *framework*, que envolveram, no entanto, alguns ajustes de pormenor em classes base. A realização destes ajustes não teve praticamente qualquer impacto ao nível da estrutura de classes, residindo as diferenças sobretudo ao nível das colaborações. Em particular ao nível dos *solvers* meta-heurísticos, há a considerar a extensão para suporte de abordagens paralelas/híbridas. Esta extensão foi já apresentada no Capítulo 5, pelo que será aqui caracterizada apenas sobre o *solver* mais básico.

Na concepção desta parte do *framework* é determinante a aplicação de dois *padrões de desenho*:

- Uma nova aplicação do *padrão* Estratégia [Gamma et al. 1995], conforme [Andreatta et al. 1998], resulta na definição de duas famílias de algoritmos, das quais o *solver* será cliente: uma de algoritmos construtivos e outra de algoritmos de pesquisa local multiobjectivo.
- A classe correspondente ao *solver* meta-heurístico é derivada de uma classe abstracta mais geral, correspondente a um processo iterativo. Esta classe constitui um *template* para processos iterativos, concretizando uma aplicação do *padrão* Método *Template*. A utilização deste *padrão* disponibiliza o suporte para criar diferentes variantes de um *solver* meta-heurístico iterativo, designadamente permitindo a extensão do *framework* a técnicas com rearranque e multiarranque.

Estrutura e participantes

O diagrama de classes apresentado na Figura B.7 apresenta a estrutura de classes para o *solver* meta-heurístico, na sua parte organizada em torno do *padrão Estratégia*.

Os participantes nesta parte do *framework* são:

- *Solver meta-heurístico* (*MetaHeuristicSolver*). Utilizará um protótipo de solução ($Solution_{Prototype}$) para criar novas soluções. Recorrerá aos serviços de algoritmos construtivos (*Constructive Algorithm*) para construir soluções para uma população inicial ($Population_{Initial}$) e uma população alargada ($Population_{Larger}$). Os serviços dos algoritmos de pesquisa local multiobjectivo (*MOLocalSearch*) permitir-lhe-ão determinar uma aproximação do conjunto de soluções eficientes (*EfficientSet*).
- *Solução* (*Solution*). No âmbito da colaboração com os algoritmos construtivos, os serviços do protótipo de solução são utilizados para criar soluções "vazias" que irão ser construídas, integradas na população inicial e na população alargada, e utilizadas nos algoritmos de pesquisa local multiobjectivo. Configura-se nesta situação uma aplicação do *padrão Protótipo*.

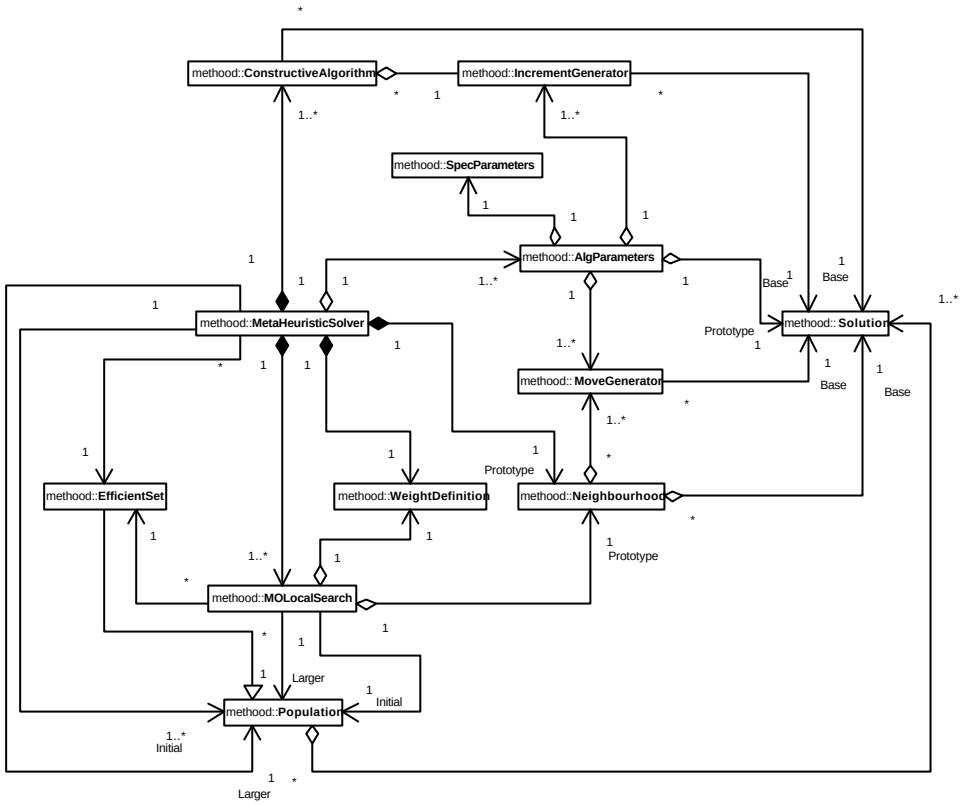


Figura B.7 Diagrama de classes para *solvers* meta-heurísticos (parte organizada sobre a aplicação do *padrão* Estratégia)

- *Algoritmo construtivo* (*ConstructiveAlgorithm*). Define o interface comum a todos os algoritmos construtivos suportados, interface este que é usado pelo *solver* para configurar e invocar um algoritmo construtivo concreto. Solicita ao gerador de incrementos (*IncrementGenerator*) com que é configurado, sucessivos incrementos, que lhe permitirão construir uma solução base ($Solution_{Base}$).
- *População inicial* (*Population*).
- *Algoritmo de pesquisa local multiobjectivo* (*MOLocalSearch*). A informação dos parâmetros do algoritmo e dos parâmetros específicos será determinante na configuração da actividade do algoritmo.
- *Vizinhança* (*Neighbourhood*).
- *Definição de pesos* (*WeightDefinition*).
- *Parâmetros do algoritmo* (*AlgParameters*).
- *Gerador de incrementos* (*IncrementGenerator*).
- *Gerador de movimentos* (*MoveGeneratorl*).
- *Parâmetros específicos dos algoritmos (SpecParameters)*.

- *Aproximação ao conjunto de soluções eficientes (EfficientSet).*

O diagrama de classes apresentado na Figura B.8 apresenta a estrutura de classes para a aplicação do *padrão* Método *Template* no *solver* meta-heurístico.

Os participantes nesta parte do *framework* são:

- *Processo Iterativo* (*IterativeProcess*).
- *Solver meta-heurístico* (*MetaHeuristicSolver*). Implementado como um processo iterativo, define as operações primitivas enumeradas na classe anterior, de acordo com as orientações apresentadas nas anotações da Figura B.8. Inclui assim, além da iteração com algoritmos construtivos e de pesquisa local, suporte explícito para pré- e pós-processamento e para a definição de algoritmos com rearranque e multiarranque. Esta definição será baseada no estabelecimento de critérios de paragem apropriados e na actualização de informação entre iterações.

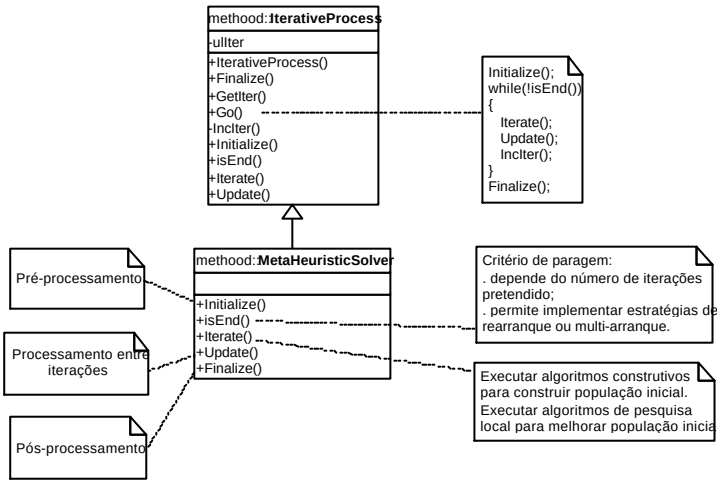


Figura B.8 Diagrama de classes para *solvers* meta-heurísticos (aplicação do *padrão* Método *Template*)

Colaborações

Na aplicação do *padrão* Método *Template*, a colaboração reside apenas em o *solver* se basear no processo iterativo para implementar o conjunto de passos invariantes.

Na parte estruturada em torno do *padrão* Estratégia, há essencialmente duas fases de colaboração entre os diversos participantes:

1. *Construção das populações inicial e alargada* (Figura B.9). Cada gerador de incrementos do conjunto que configura o *solver* corresponde a uma das soluções que irá integrar a população inicial. Desta forma se especifica o número de soluções da população inicial e o respectivo algoritmo de construção. As soluções são criadas através dos serviços da solução protótipo, numa aplicação do *padrão Protótipo*.

Para cada solução é, então, criado um algoritmo construtivo, configurado com a solução e o gerador de incrementos correspondente. A esse algoritmo é solicitada a construção da solução, posteriormente integrada nas populações inicial e alargada.

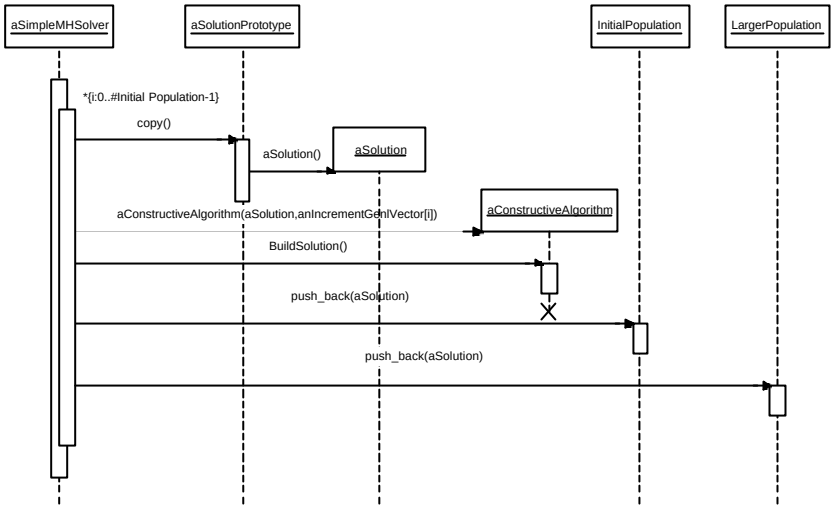


Figura B.9 Diagrama de sequência para a construção das populações inicial e alargada em *solvers* meta-heurísticos

2. *Pesquisa local multiobjectivo* (Figura B.10). Preliminarmente é criado um conjunto de objectos que irão configurar a actividade do algoritmo: uma vizinhança, configurada com o primeiro (e único, no caso de uma vizinhança simples) gerador de movimentos do conjunto que configura o *solver*; uma estratégia de definição de pesos; um conjunto de componentes particulares ao algoritmo concreto que se irá utilizar, de acordo com os parâmetros específicos dos algoritmos. Será, então, criado um algoritmo de pesquisa local multiobjectivo, configurado com todos estes elementos e, ainda, com as populações inicial e alargada e com a aproximação ao conjunto de soluções eficientes. Este algoritmo actualizará a aproximação ao conjunto de soluções eficientes com a população inicial e com as diversas soluções encontradas ao longo da sua pesquisa.

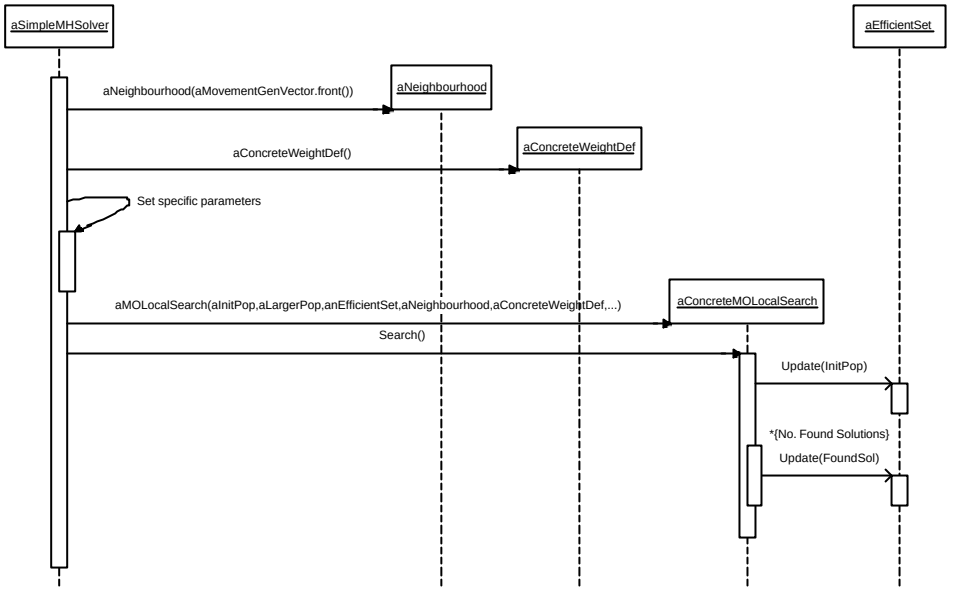


Figura B.10 Diagrama de sequência para a interacção com pesquisa local multiobjectivo em *solvers* meta-heurísticos

B.6 Algoritmos construtivos

De entre as diversas abordagens OO apresentadas no Capítulo 4, apenas o *framework Searcher* [Andreatta et al. 1998] disponibiliza um suporte explícito para algoritmos construtivos. Um conceito central no desenho dessa parte do *framework* é o de *incremento.* É também em torno deste conceito que se estrutura a parte do METHOOD relativa aos algoritmos construtivos.

Para clarificação da ideia, será útil adoptar uma perspectiva de uma solução como um conjunto de componentes mais elementares. Um incremento consistirá, então, em acrescentar a uma solução um destes seus componentes mais elementares.

Com base neste conceito, os algoritmos construtivos podem ser modelados como realizações de sucessivos incrementos sobre uma solução, inicialmente *vazia*, até esta ficar *completa*.

A concepção desta parte do *framework* assenta na aplicação de dois *padrões de desenho*:

- O *padrão* Protótipo. A cada gerador de incrementos está associado um protótipo de incremento, cujos serviços são usados para criar novos incrementos.
- O *padrão* Iterador. Um gerador de incrementos pode ser visto como um contentor de incrementos, à semelhança da relação que se estabeleceu, no âmbito da pesquisa local multiobjectivo, entre movimentos e vizinhanças ou geradores de movimento.

Estrutura e participantes

O diagrama de classes apresentado na Figura B.11 apresenta a estrutura de classes para os algoritmos construtivos.

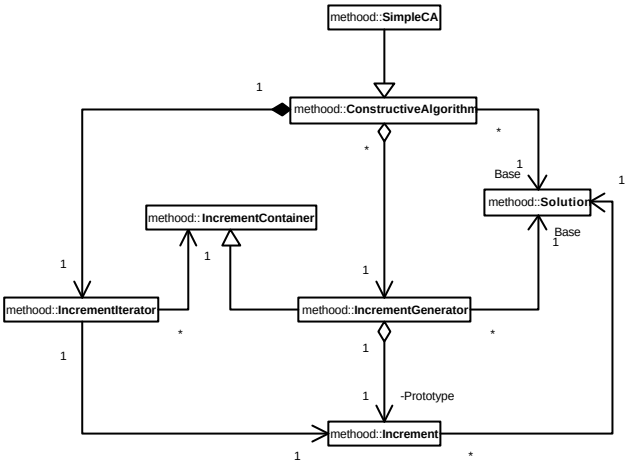


Figura B.11 Diagrama de classes para algoritmos construtivos

Os participantes nesta parte do *framework* são:

- *Algoritmo construtivo* (*ConstructiveAlgorithm*). Para a interacção com o gerador de incrementos, o algoritmo construtivo utiliza os serviços de um iterador de incrementos (*IncrementIterator*). Para tal, o gerador de incrementos implementa o interface definido pelo contentor de incrementos (*IncrementContainer*).
- *Algoritmo construtivo simples* (*SimpleCA*). Implementa um algoritmo construtivo concreto. A sua caracterização como simples deriva do facto de utilizar o iterador de incrementos para meramente solicitar, um a um, os sucessivos incrementos que permitirão construir uma solução. Um exemplo de implementação mais complexa, não considerada nesta versão do *framework*, é o caso dos algoritmos construtivos do GRASP. Orientações para este caso são apresentadas após a descrição das colaborações.
- *Iterador de incrementos* (*IncrementIterator*). Implementa o acesso sequencial aos elementos de um contentor de incrementos e mantém a posição actual da travessia do contentor.
- *Contentor de incrementos* (*IncrementContainer*). Define o interface que um contentor concreto deve disponibilizar para acesso por um iterador.

- *Incremento* (*Increment*). Os serviços do protótipo de incremento (*Increment*$_{Prototype}$) são utilizados para criar incrementos "vazios" que irão ser configurados pelo gerador de incrementos. O incremento disponibiliza um serviço para a sua própria execução sobre uma solução base.
- Gerador de incrementos (*IncrementGenerator*).
- Solução (*Solution*).

Colaborações

As colaborações existentes nesta parte do *framework* serão apresentadas com base no caso concreto do algoritmo construtivo simples (Figura B.12).

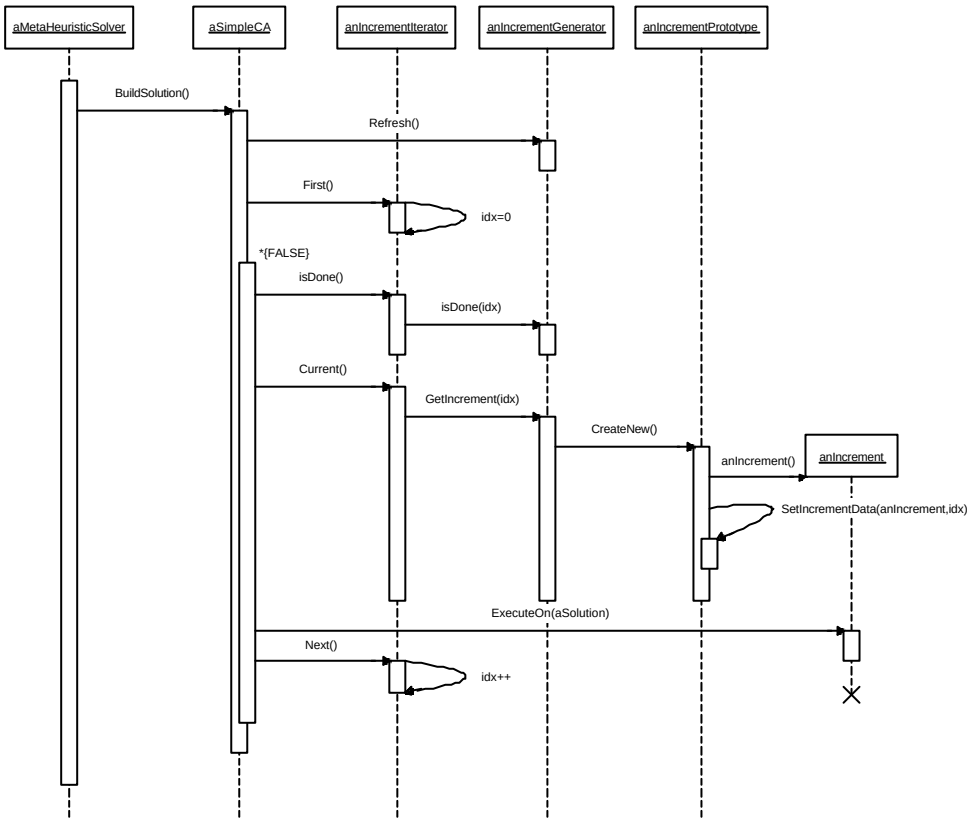


Figura B.12 Diagrama de sequência para a construção de soluções

O algoritmo construtivo solicita inicialmente ao gerador de incrementos um *refresh*. Este serviço destina-se a permitir actualizações internas de informação do gerador de incrementos como, por exemplo, a geração de um novo conjunto aleatório de incrementos, no caso de geradores aleatórios.

O iterador irá, então, percorrer todos os incrementos, sendo cada um destes executado sobre a solução base. A sequência do incremento é identificada por um índice. À obtenção de cada incremento está associada a criação de uma cópia do incremento, realizada com recurso aos serviços do protótipo de incremento, que cria um incremento e o disponibiliza ao gerador para configuração adequada.

Sobre a estrutura de classes apresentada, é possível, com esquemas de colaboração alternativos, implementar outro tipo de algoritmos. Em particular, é possível implementar o tipo de algoritmos construtivos utilizados no *GRASP*. Tal exige uma utilização diferente do iterador e do gerador de incrementos: este último, em vez de conter a sequência de incrementos para construir uma solução, irá conter, em cada passo da construção da solução, e de acordo com o estado desta, as alternativas aceitáveis de incrementos para esse passo; o iterador será utilizado para percorrer um conjunto de alternativas para cada passo, e não, como no caso do algoritmo simples, os incrementos sucessivos que constróem completamente uma solução.

Um algoritmo construtivo realizará então um número de iterações igual ao número de incrementos necessários para completar uma solução. Em cada iteração, o iterador de incrementos permitirá navegar pelas diversas alternativas geradas pelo gerador. Entre essas, e de acordo com um critério determinado (*greedy*, aleatório e adaptativo no caso do *GRASP*), o algoritmo seleccionará uma, que será executada, passando-se então à iteração seguinte.

C
INDEXAÇÃO DE VIZINHANÇAS

C.1 Indexação de vizinhanças do tipo *swap*

Sejam 1, 2, ..., *n* as posições possíveis para as tarefas e defina-se que, de modo a evitar repetições, a tarefa da posição *p* (com *p* entre 1 e *n* - 1) apenas poderá trocar de posição com as *n* - *p* tarefas subsequentes.

Seja o número total das trocas para todas as tarefas entre as posições 1 e *p*, dado por

$$S(p)=\frac{(n-1+n-p)p}{2}.$$

Esta função é monótona crescente, para *p* entre 1 e *n* - 1, sendo-o igualmente a sua inversa

$$S^{-1}(i)=\frac{(2n+1)-\sqrt{(2n+1)^2-8(n+i)}}{2},$$

para *i* entre 0 e $n(n-1)/2-1$, em que $n(n-1)/2$ é o número total de trocas para todas as tarefas.

Seja um índice *i* que corresponde univocamente a cada uma das trocas possíveis, tal que *i*=0 corresponde à troca das tarefas 1 e 2, *i*=1 das tarefas 1 e 3, ..., *i*=*n-2* das tarefas 1 e *n*, *i*=*n*-1 das tarefas 2 e 3, e assim sucessivamente.

O valor do índice correspondente à troca entre as tarefas p_{i1} e p_{i2}, com $p_{i2} > p_{i1}$, é, assim, dado pela expressão

$$i=S(p_{i1}-1)+(p_{i2}-p_{i1}-1).$$

Sejam os índices *i'* e *i''*, correspondentes às trocas entre p_{i1} e $p_{i1}+1$, e $p_{i1}+1$ e $p_{i1}+2$, respectivamente. Tem-se, para estes índices, que $i'=S(p_{i1}-1)$ e $i''=S(p_{i1})$. Sejam, ainda, o índice *j*, tal que $i'< j \le i''$, e o valor real *q*, tal que $q-1=S^{-1}(j)$.

De $i'< j \le i''$, como S^{-1} é monótona crescente, vem $S^{-1}(i')< S^{-1}(j)< S^{-1}(i'')$, ou seja, $p_{i1}-1< q-1< p_{i1}$, resultando que $p_{i1}=\text{int}(q)$, em que **int** é um operador que permite obter a parte inteira do operando.

p_{i1} é, então, dada por

$$p_{i1} = \text{int}\left(\frac{(2n+1) - \sqrt{(2n+1)^2 - 8(n+i)}}{2} \right).$$

Para p_{i2} a expressão será:

$$p_{i2} = p_{i1} + i - S(p_{i1} - 1) + 1 \ .$$

C.2 Indexação de vizinhanças do tipo *shift*

Sejam 1, 2, ..., n as posições possíveis para as tarefas, e defina-se que a tarefa da posição p (com p entre 1 e n - 1) poderá ser deslocada para qualquer posição q, desde que $p \neq q$ e $q \neq p$-1, de modo a evitar repetições.

Nestas condições, a tarefa da posição 1 poderá ser deslocada para n - 1 posições, enquanto todas as outras apenas poderão ser deslocadas para n - 2 posições.

Seja, ainda, um índice i que corresponde univocamente a cada um dos deslocamentos possíveis, tal que i=0 corresponde a deslocar a tarefa da posição 1 para a 2, i=1 da posição 1 para a 3, ..., i=n-2 da posição 1 para a n, i=n-1 da posição 2 para a 3, e assim sucessivamente.

O índice i fica assim associado às posições de deslocamento p_i e q_i da seguinte forma:

$$p_i = \begin{cases} 1 & \text{se } i \leq n-2 \\ \text{int}\left(\dfrac{i-(n-1)}{n-2} \right) + 2 & \text{se } i > n-2 \end{cases}$$

$$q_i = \begin{cases} i+2 & \text{se } i \leq n-2 \\ q'_i & \text{se } i > n-2 \end{cases},$$

com

$$q'_i = \begin{cases} \text{mod}\left(\dfrac{i-(n-1)}{n-2} \right) + 1 & \text{se mod}\left(\dfrac{i-(n-1)}{n-2} \right) + 1 < p_i - 1 \\ \text{mod}\left(\dfrac{i-(n-1)}{n-2} \right) + 1 + 2 & \text{se mod}\left(\dfrac{i-(n-1)}{n-2} \right) + 1 \geq p_i - 1 \end{cases},$$

em que **mod** é um operador que permite obter o resto da divisão inteira do numerador pelo denominador do operando.

D
DISTRIBUIÇÃO DAS SOLUÇÕES DOS CONJUNTOS DE REFERÊNCIA

Conforme referido a propósito das medidas de desempenho a utilizar nos testes computacionais realizados (Capítulo 7), a utilização com confiança da medida de distância *D*1 (7.1) pressupõe uma distribuição uniforme das soluções do conjunto de referência [Hansen, Jaskiewicz 1998].

A verificação desta uniformidade foi realizada, neste trabalho, com base nos mecanismos de *"fitness sharing"*, utilizados nos GA multiobjectivo. Para cada solução do conjunto de referência é definido um nicho, dentro do qual é realizada uma contagem de soluções, a designar por contagem de nicho (*"niche count"*), contagem esta que tem em conta não só o número de soluções, mas também a sua proximidade. Verifica-se, então, para cada conjunto de referência, se, para alguma solução este valor é muito diferente dos valores das restantes soluções.

A definição da dimensão do nicho e da contagem de nicho são realizadas com base no trabalho de [Fonseca, Fleming 1993], que utilizam *"fitness sharing"* no espaço dos objectivos, no âmbito de GA multiobjectivo. Neste trabalho faz-se a sua adaptação para a utilização de factores de equalização de gamas, em problemas com dois e três objectivos.

A dimensão do nicho, s_{share}, é obtida a partir da seguinte equação:

$$N = \frac{\prod_{i=1}^{r}\left(f'_{i,\max} - f'_{i,\min} + 2s_{share}\right) - \prod_{i=1}^{r}\left(f'_{i,\max} - f'_{i,\min}\right)}{\left(2s_{share}\right)^{r}}, \qquad \text{(D.1)}$$

em que N é a cardinalidade do conjunto de soluções, r é o número de objectivos e $f'_{i,\max}$ e $f'_{i,\min}$ são, respectivamente, os valores máximo e mínimo do objectivo i no conjunto de soluções. $f'_{i,\max} - f'_{i,\min}$ será, portanto, a gama de valores do objectivo *i*.

Ao considerar as gamas equalizadas, a expressão anterior é consideravelmente simplificada, obtendo-se

$$N = \frac{\left(R_{eq} + 2s_{share}\right)^{r} - {R_{eq}}^{r}}{\left(2s_{share}\right)^{r}}. \qquad \text{(D.2)}$$

Para dois objectivos (r = 2), σ_{share} é dada por

$$\sigma_{share} = R_{eq} \frac{2}{N-2}, \tag{D.3}$$

excluindo-se a raiz nula.

Para três objectivos (r = 3), σ_{share} é dada por

$$\sigma_{share} = R_{eq} \frac{1+\sqrt{1+\frac{4}{3}(N-1)}}{\frac{4}{3}(N-1)}, \tag{D.4}$$

excluindo-se as raízes nula e negativa.

A contagem de nicho, para uma solução j, nc_j, é dada por

$$nc_j = \sum_k \max\left(0, 1 - \frac{d\left(f'(x_j), f'(x_k)\right)}{\sigma_{share}}\right), \tag{D.5}$$

em que $\sum_k$ é o somatório para todas as soluções do conjunto, e $d\left(f'(x_j), f'(x_k)\right)$ é uma métrica Euclidiana (L_2) no espaço dos objectivos, com as diferenças de valores nos objectivos individuais afectadas dos correspondentes factores de equalização de gamas.

Para o exemplo apresentado em [Hansen, Jaskiewicz 1998], com as soluções do conjunto de referência apresentadas na Tabela D.1, com N = 5, obtém-se para R_{eq} o valor de 2.505 e para σ_{share} o valor de 1.67. Os respectivos valores de contagem de nicho são igualmente apresentados na Tabela D.1, em valor absoluto e em percentagem do máximo valor possível para uma contagem de nicho, ou seja, a própria cardinalidade do conjunto, N.

Solução	nc_j	nc_j / N (%)
[9, 4]	1	20
[4, 9]	2.992	59.83
[6, 6]	1	20
[3.99, 9.01]	2.987	59.75
[4.01, 8.99]	2.987	59.75

Tabela D.1 Contagens de nicho para o exemplo de [Hansen, Jaskiewicz 1998]

Verifica-se um desnível elevado entre os valores mínimo e máximo das contagens, de 39.83%, correspondente a uma não uniformidade da distribuição das soluções neste conjunto.

Para os conjuntos de referência considerados neste trabalho, realizou-se uma análise idêntica, mas para uma situação de três objectivos. Na Tabela D.2 apresenta-se, para os conjuntos de referência de cada instância, os valores de N, R_{eq}, σ_{share}, e os valores mínimos,

médios e máximos das contagens de nicho, quer em valores absolutos, quer em valores relativos. A composição dos conjuntos de referência é apresentada no Anexo E.

Instância	N	R_{eq}	s_{share}	nc_j			nc_j/N (%)			
				Mín.	Méd.	Máx.	Mín.	Méd.	Máx.	Amplitude
2	37	2.971	0.495	1.000	1.990	3.104	2.70	5.38	8.39	5.69
3	7	1.980	0.990	1.000	1.461	2.077	14.29	20.86	29.68	15.39
4	13	2.949	0.944	1.000	1.403	2.311	7.69	10.79	17.77	10.08
5	15	0.992	0.289	1.000	1.851	2.680	6.67	12.34	17.87	11.20
6	67	4.798	0.569	1.000	1.693	2.732	1.49	2.53	4.08	2.58
7	55	2.966	0.393	1.000	1.923	2.787	1.82	3.50	5.07	3.25
8	63	3.946	0.484	1.000	1.933	3.937	1.59	3.07	6.25	4.66
9	32	2.958	0.537	1.000	1.922	2.835	3.23	6.20	9.15	5.92
10	99	3.963	0.378	1.000	2.053	3.231	1.02	2.10	3.30	2.28

Tabela D.2 Contagens de nicho para os conjuntos de referência

Não se verifica, para qualquer dos conjuntos de referência considerados, uma diferença entre o mínimo e o máximo valores das contagens tão significativa como, por exemplo, a obtida no caso referido em [Hansen, Jaskiewicz 1998], sendo a maior diferença registada apenas de 15.39%. Na presença destes níveis de valores, considera-se razoável assumir que o pressuposto de uniformidade da distribuição das soluções se verifica para todos os conjuntos de referência considerados, pelo que a distância $D1$ poderá ser utilizada com confiança.

E
COMPOSIÇÃO DOS CONJUNTOS DE REFERÊNCIA

Instância	Objectivo	Soluções									
WT040002	$\Sigma w_j T_j$	1225	1229	1263	1267	1306	1451	1476	1502	1513	1527
	ΣU_j	5	5	4	4	5	5	4	6	3	3
	L_{max}	352	300	352	300	281	270	281	264	583	531
	$\Sigma w_j T_j$	1556	1570	1586	1590	1618	1620	1652	1676	1704	1747
	ΣU_j	6	5	3	3	3	6	3	4	4	4
	L_{max}	251	251	372	352	329	234	327	270	267	264
	$\Sigma w_j T_j$	1797	1803	1829	1927	1936	1990	2024	2088	2186	2195
	ΣU_j	5	5	6	6	4	5	6	6	3	4
	L_{max}	248	234	231	226	253	220	218	213	302	250
	$\Sigma w_j T_j$	2221	2363	2402	2435	2587	2803	2983			
	ΣU_j	3	3	4	5	3	4	4			
	L_{max}	289	286	241	213	272	239	230			
WT040003	$\Sigma w_j T_j$	537	549	573	585	716	1432	2658			
	ΣU_j	4	4	3	3	3	4	2			
	L_{max}	288	212	288	212	193	187	402			
WT040004	$\Sigma w_j T_j$	2094	2100	2154	2175	2181	2399	2829	3021	3090	3299
	ΣU_j	3	3	3	6	4	3	3	5	4	4
	L_{max}	393	319	261	236	236	257	236	210	225	221
	$\Sigma w_j T_j$	3549	3871	4348							
	ΣU_j	4	6	5							
	L_{max}	210	206	206							

WT040005	$\Sigma w_j T_j$	990	1038	1080	1090	1180	1394	1470	1484	1874	1894
	ΣU_j	4	4	4	3	3	4	4	4	4	3
	L_{max}	352	326	325	352	325	324	298	297	272	272
	$\Sigma w_j T_j$	1920	1984	2078	2124	2935					
	ΣU_j	4	3	4	4	3					
	L_{max}	246	245	244	218	218					
WT040006	$\Sigma w_j T_j$	6955	7019	7071	7080	7135	7153	7169	7196	7235	7269
	ΣU_j	11	11	10	11	10	11	11	10	11	10
	L_{max}	711	690	711	679	690	670	658	679	652	670
	$\Sigma w_j T_j$	7285	7308	7351	7424	7470	7534	7552	7595	7668	7668
	ΣU_j	10	11	10	10	9	9	11	9	9	10
	L_{max}	658	643	652	643	711	690	631	679	670	631
	$\Sigma w_j T_j$	7684	7750	7823	8067	8111	8133	8138	8143	8211	8216
	ΣU_j	9	9	9	9	11	11	11	8	11	8
	L_{max}	658	652	643	631	621	618	615	711	606	702
	$\Sigma w_j T_j$	8233	8294	8367	8436	8455	8458	8463	8536	8553	8611
	ΣU_j	8	8	8	10	11	10	10	10	11	8
	L_{max}	690	679	670	621	598	618	615	606	594	658
	$\Sigma w_j T_j$	8780	8878	8938	9070	9141	9143	9155	9360	9595	9651
	ΣU_j	10	10	9	9	12	9	9	11	10	8
	L_{max}	598	594	621	619	591	618	598	591	591	653
	$\Sigma w_j T_j$	9658	9731	9975	10080	10615	10688	10788	11693	11872	12389
	ΣU_j	8	8	8	9	8	8	8	8	8	7
	L_{max}	651	642	630	591	619	618	610	605	593	711
	$\Sigma w_j T_j$	12462	12641	12680	15115	15188	15261	17299			
	ΣU_j	7	7	8	7	7	7	7			
	L_{max}	702	690	591	651	642	624	610			

WT040007	$\Sigma w_j T_j$	6324	6331	6375	6511	6518	6525	6562	6569	6712	6756
	ΣU_j	9	9	9	8	8	9	8	9	8	8
	L_{max}	884	852	839	884	852	823	839	810	823	810
	$\Sigma w_j T_j$	6954	6961	7005	7141	7147	7148	7191	7192	7334	7378
	ΣU_j	9	9	9	8	9	8	9	8	8	8
	L_{max}	805	773	760	805	744	773	731	760	744	731
	$\Sigma w_j T_j$	7826	7833	7996	8003	8013	8020	8183	8190	8266	8273
	ΣU_j	9	9	9	9	8	8	8	8	7	7
	L_{max}	705	682	676	653	705	682	676	653	884	852
	$\Sigma w_j T_j$	8317	8888	8895	8939	9305	9392	9399	9599	9671	9744
	ΣU_j	7	7	7	7	9	9	9	8	9	7
	L_{max}	839	805	773	760	642	636	619	642	613	705
	$\Sigma w_j T_j$	9751	10102	10224	11007	11020	11152	11421	11421	11626	12198
	ΣU_j	7	8	9	8	8	10	8	9	8	8
	L_{max}	682	640	612	637	619	611	618	611	613	612
	$\Sigma w_j T_j$	12421	12625	13007	13414	13421					
	ΣU_j	7	8	7	7	7					
	L_{max}	679	611	637	630	618					
WT040008	$\Sigma w_j T_j$	6865	7097	7125	7142	7240	7408	7442	7500	7508	7696
	ΣU_j	9	9	8	9	9	8	9	8	9	11
	L_{max}	876	855	876	827	783	827	762	783	734	713
	$\Sigma w_j T_j$	7697	7710	7756	7777	7882	7883	8012	8059	8098	8147
	ΣU_j	10	9	8	7	11	10	7	8	7	9
	L_{max}	713	713	734	876	692	692	827	713	783	692
	$\Sigma w_j T_j$	8242	8324	8510	8719	9079	9110	9116	9218	9262	9300
	ΣU_j	11	7	10	10	8	11	9	8	7	9
	L_{max}	688	734	688	682	687	678	672	666	687	658

	$\Sigma w_j T_j$	9304	9411	9459	9488	9510	9511	9566	9573	9591	9632
	ΣU_j	10	9	9	7	11	10	9	9	8	11
	L_{max}	651	645	638	638	630	630	630	617	630	609
	$\Sigma w_j T_j$	9633	9653	9679	9687	9998	10451	10701	10713	10725	10726
	ΣU_j	10	8	9	9	10	10	8	9	7	7
	L_{max}	609	617	609	596	592	586	596	586	623	618
	$\Sigma w_j T_j$	10727	10791	10855	10883	10900	10901	10965	11029	11057	11089
	ΣU_j	9	10	10	8	7	9	9	9	8	9
	L_{max}	576	571	568	576	604	571	564	562	571	560
	$\Sigma w_j T_j$	11121	11185	11193	11194						
	ΣU_j	8	8	8	7						
	L_{max}	564	562	560	560						
WT040009	$\Sigma w_j T_j$	16225	16270	16287	16332	16439	16484	16539	16601	16753	17041
	ΣU_j	10	9	10	9	10	9	8	8	8	11
	L_{max}	892	892	808	808	793	793	892	808	793	781
	$\Sigma w_j T_j$	17055	17062	17135	17180	17425	17450	17523	17568	17813	17929
	ΣU_j	10	11	10	9	8	11	10	9	8	11
	L_{max}	781	726	726	726	726	713	713	713	713	699
	$\Sigma w_j T_j$	18101	18317	18373	18489	18761	19613	19992	20231	20278	20561
	ΣU_j	10	11	9	10	9	8	11	10	8	9
	L_{max}	699	686	699	686	686	712	685	685	692	685
	$\Sigma w_j T_j$	28421									
	ΣU_j	8									
	L_{max}	686									
WT040010	$\Sigma w_j T_j$	9737	9741	9755	9759	9783	9787	9793	9797	9825	9829
	ΣU_j	12	11	12	11	12	11	12	11	12	11
	L_{max}	1131	1131	1053	1053	1001	1001	985	985	933	933

$\Sigma w_j T_j$	9839	9843	9895	9899	9941	9945	9955	9959	10013	10031
ΣU_j	12	11	12	11	12	11	12	11	10	10
L_{max}	921	921	905	905	892	892	887	887	1131	1053
$\Sigma w_j T_j$	10059	10069	10101	10115	10171	10217	10231	10374	10392	10420
ΣU_j	10	10	10	10	10	10	10	9	9	9
L_{max}	1001	985	933	921	905	892	887	1131	1053	1001
$\Sigma w_j T_j$	10430	10462	10476	10585	10743	10757	10971	10975	10985	10989
ΣU_j	9	9	9	9	9	9	12	11	12	11
L_{max}	985	933	921	905	892	887	851	851	839	839
$\Sigma w_j T_j$	11041	11045	11087	11091	11348	11362	11418	11464	11565	11579
ΣU_j	12	11	12	11	10	10	10	10	9	9
L_{max}	823	823	815	815	851	839	823	815	851	839
$\Sigma w_j T_j$	11688	12144	13489	13493	13497	13503	13507	13515	13559	13563
ΣU_j	9	9	12	11	8	12	11	8	12	11
L_{max}	823	815	780	780	1131	768	768	1053	752	752
$\Sigma w_j T_j$	13605	13609	13624	13847	13861	13917	13955	13963	13969	14078
ΣU_j	12	11	8	10	10	10	9	10	9	9
L_{max}	744	744	1001	780	768	752	780	744	768	752
$\Sigma w_j T_j$	14373	14482	14531	14779	14833	14837	15175	15506	15615	15936
ΣU_j	8	8	9	8	12	11	10	8	8	9
L_{max}	971	919	744	904	738	738	738	874	822	738
$\Sigma w_j T_j$	16194	16249	16359	16713	17093	17167	17199	17411	17517	18138
ΣU_j	11	10	9	8	11	8	11	10	10	9
L_{max}	724	724	724	821	716	815	705	716	705	718
$\Sigma w_j T_j$	20432	20521	21079	21257	21553	21659	22344	24718		
ΣU_j	8	8	8	8	9	9	8	8		
L_{max}	799	791	752	750	716	705	744	738		

Referências

[Aarts, Lenstra 1995] Aarts, E. H. L. e Lenstra, J. K. (eds.) - *Local Search in Combinatorial Oprimization* - Wiley, Chichester, 1995.

[Abdul-Razaq et al. 1990] Abdul-Razaq, T. S.; Potts, C. N. e van Wassenhove, L. N. - *A survey of algorithms for the single-machine total weighted tardiness scheduling problem* - Discrete Applied Mathematics, 26, pp. 235-253, 1990.

[Aggoun et al. 1995] Aggoun, A.; Chan, D.; ...; de Villeneuve, D. H. - *ECL^{i}PS^e 3.5: ECRC Common Logic Programming System: User Manual* - European Computer-Industry Research Centre, 1995.

[Akturk, Yildirim 1998] Akturk, M. S. e Yildirim, M. B. - *A new lower bounding scheme for the total weighted tardiness problem* - Computers and Operations Research, Vol. 25, No. 4, pp. 265-278, 1998.

[Almeida, Centeno 1998] Almeida, M. T. e Centeno, M. - *A composite heuristic for the single machine early/tardy job scheduling problem* - Computers and Operations Research, Vol. 25, No. 7/8, pp. 625-635, 1998.

[Andreatta et al. 1998] Andreatta, A.; Carvalho, S. e Ribeiro, C. - *An Object-Oriented Framework for Local Search Heuristics* - 26th Conference on Technology of Object-Oriented Languages and Systems (TOOLS USA'98), IEEE Press, pp. 33-45, 1998.

[Antunes et al. 1993] Antunes, C. H.; Craveirinha, J. F. e Clímaco, J. N. - *A Multiple Criteria Model for New Telecommunication Service Planning* - European Journal of Operational Research, Vol. 71, pp. 341-352, 1993.

[Alexander 1977] Alexander, C.; Ishikawa, S.; Silverstein, M.; Jacobsen, M.; Fiksdahl-King, I. e Angel, S. - *A Pattern Language* - Oxford University Press, 1977.

[Baker 1974] Baker, K. R. - *Introduction to Sequencing and Scheduling* - John Wiley, New York, 1974.

[Baker, Scudder 1990] Baker, K. R. e Scudder, G. D. - *Sequencing with Earliness and Tardiness Penalties: A review* - Operations Research, 38, pp. 22-36, 1990.

[Barr et al. 1995] Barr, R. S.; Golden, B. L.; Kelly, J. P.; Resende, M . G. e Stewart, W. R. - *Designing and Reporting on Computational Experiments with Heuristic Methods* - Journal of Heuristics, 1, pp. 1-32, 1995.

[Bauer et al. 1999] Bauer, A.; Bullnheimer, B.; Hartl, R. F. e Strauss, C. - *An ant colony optimization approach for the single machine total tardiness problem* - Proceedings of CEC'99, pp. 1445-1450, IEEE Press, Piscataway, NK - 1999.

[Beasley 1990] Beasley, J. E. - *OR-Library: Distributing test problems by electronic mail* - Journal of the Operational Research Society, 41, pp. 1069-1072, 1990.

[Beasley 1999] Beasley, J. E. - *Population Heuristics* - Working paper, The Management School, Imperial College, London, England, 1999.

[Beldiceanu, Contejean 1994] Beldiceanu, N. e Contejean, E. - *Introducing Global Constraints in CHIP* - Elsevier Science, editor, Mathematical Computation Modelling, Vol. 20, pp. 97-123, 1994.

[Belton, Elder 1996] Belton, V. e Elder, M. D. - *Exploring a Multicriteria Approach to Production Scheduling* - Journal of the Operational Research Society, Vol. 47, pp. 162-174, 1996.

[Blazewicz et al. 1986] Blazewicz, J.; Cellary, W.; Slowinski, R. e Weglarz, J. - *Scheduling under Resource Constraints - Deterministic Models* - Annals of Operations Research, Vol. 7, Baltzer, Basel, 1986.

[Blazewicz et al. 1991] Blazewicz, J.; Dror, M. e Weglarz, J. - *Mathematical programming formulations for machine scheduling: a survey* - European Journal of Operational Research, Vol. 51, pp. 283-300, 1991.

[Blazewicz et al. 1993] Blazewicz, J.; Ecker, K.; Schmidt, G. e Weglarz, J. - *Scheduling in Computer and Manufacturing Systems* - Springer Verlag, Berlin, 1993.

[Bock 1958] Bock, F. - *An Algorithm for Solving 'Traveling Salesman' and Related Network Optimization Problems* - Comunicação no 14th ORSA Meeting, St. Louis, 24 de Outubro de 1958.

[Booch 1994] Booch, G. - *Object-Oriented Analysis and Design with Applications* - 2nd Ed., Benjamin Cummings, 1994.

[Buschmann, Meunier 1994] Buschmann, F. e Meunier, R. - *A System of Patterns* - Proceedings of the First Conference on Pattern Languages and Programming, Addison-Wesley, 1994.

[Charon, Hudry 1993] Charon, I. e Hudry, O. - *The Noising Method: a New Combinatorial Optimization Method* - Operations Research Letters, Vol. 14, pp. 133-137, 1993.

[Charon, Hudry 1999] Charon, I. e Hudry, O. - *Principles and Implementations of the Noisy Methods* - Resumo alongado, Third Metaheuristics International Conference, Angra dos Reis, Brasil, Julho 19-23, 1999.

[Chen, Bulfin 1993] Chen, C.-L. e Bulfin, R. L. - *Single machine multi-criteria scheduling* - European Journal of Operation Research, 70, pp. 115-125, 1993.

[Coad 1992] Coad, P. - *Object-Oriented Patterns* - Communications of the ACM, Vol. 35, No. 9, 1992.

[Coello 1999] Coello, C. A. C. - *A Comprehensive Survey of Evolutionary-based Multiobjective Optimization Techniques* - Knowledge and Information Systems. An International Journal, Vol. 1, No. 3, pp. 269-308, 1999.

[Colorni et al. 1991] Colorni, A.; Dorigo, M. e Maniezzo, V. - *Distributed Optimization by Ant Colonies* - Proceedings of European Conference on Artificial Life ECAL'91, Paris, França, Elsevier Publishing, pp. 134-142, 1991.

[Colorni et al. 1996] Colorni, A.; Dorigo, M.; Maffioli, F. ; Maniezzo, V.; Righini, G. e Trubian, M. - *Heuristics from Nature for Hard Combinatorial Optimization Problems* - International Transactions in Operational Research, Vol. 3, No. 1, pp. 1-21, 1996.

[Cottrell, Fort 1987] Cottrell, M. e Fort, J. C. - *Étude d'un Processus d'Autoorganisation* - Annales de l'Institut Poincarré, Vol. 23, pp. 1-20, 1987.

[Crawels et al. 1998] Crawels, H. A. J.; Potts, C. N. e Van Wassenhove, L. N. - *Local Search Heuristics for the Single Machine Total Weighted Tardiness Scheduling Problem* - INFORMS Journal on Computing, Vol. 10, No. 3, pp. 341-350, 1998.

[Croes 1958] Croes, G. A. - *A Method for Solving Traveling Salesman Problems* - Operations Research, Vol. 6, pp. 791-812, 1958.

[Cunnigham, Smyth 1995] Cunningham, P. e Smyth, B. - *CBR in scheduling: reusing solution components* - Comunicação no 14th Workshop of the UK Planning and Scheduling Special Interest Group, 1995.

[Current, Min 1986] Current, J. e Min, H. - *Multiobjective Design of Transportation Networks: Taxonomy and Annotation* - European Journal of Operational Research, Vol. 26, pp. 187-201, 1986.

[Czyzac, Jaszkiewicz 1998] Czyzac, P. e Jaszkiewicz, A. - *Pareto Simulated Annealing - a Metaheuristic Technique for Multiple Objective Combinatorial Optimization* - Journal of Multicriteria Decision Analysis, 7, pp. 34-47, 1998.

[Diáz et al. 1996] Diáz, Adenso; Glover, Fred; Ghaziri, Hassan; Gonzalez, J. L.; Laguna, Manuel; Moscato, Pablo e Tseng, Fan T. - *Optimización Heurística y Redes Neuronales en Dirección de Operaciones e Ingeniería* - Editorial Paraninfo, Madrid, 1996.

[Dorigo, Di Caro 1999] Dorigo, M. e Di Caro, G. - *The Ant Colony Optimization Metaheuristic* - Corne, D.; Dorigo, M. e Glover, F. (eds.), New Ideas in Optimization, McGraw-Hill (em publicação), 1999.

[Dueck, Scheuer 1990] Dueck, G. e Scheuer, T. - *Threshold Accepting. A General Purpose Optimization Algorithm Appearing Supperior to Simulated Annealing* - Journal of Computational Physics, Vol. 90, pp. 161, 1990.

[Durbin, Wilshaw 1987] Durbin, R. e Wilshaw, D. - *An Analogue Approach to the Travelling Salesman Problem Using an Elastic Net Method* - Nature, Vol. 326, pp. 689-691, 1987.

[Ehrgott 2000] Ehrgott, M. - *Approximation algorithms for combinatorial multicriteria optimization problems* - International Transactions in Operational Research, Vol. 7, No. 1, pp. 5-31, 2000.

[Ehrgott, Gandibleux 2000] Ehrgott, M. e Gandibleux, X. - *An Annotated Bibliography of Multiobjective Combinatorial Optimization* - Report in Wirtschaftsmathematik, Nr 62/2000, Fachbereich Mathematik - Universitat Kaiserslautern, 2000.

[Emmons 1969] Emmons, H. - *One machine sequencing to minimize certain functions of job tardiness* - Operations Research, 17, pp. 701-715, 1969.

[Feo, Resende 1989] Feo, T. A. e Resende, M. G. C. - *A Probabilistic Heuristic for a Computationally Difficult Set Covering Problem* - Operations Research Letters, Vol. 8, pp. 67-71, 1989.

[Feo, Resende 1995] Feo, T. A. e Resende, M. G. C. - *Greedy Randomized Adaptive Search Procedures* - Journal of Global Optimization - Vol. 6, pp. 109-133, 1995.

[Feo et al. 1996] Feo, T. A.; Sarathy, K. e McGahan, J. - *A GRASP for single machine scheduling with sequence dependent costs and linear delay penalties* - Computers and Operations Research, Vol. 23, No. 9, pp. 881-895, 1996.

[Ferland et al. 1996] Ferland, J.; Hertz, A. e Lavoie, A. - *An Object-Oriented Methodology for Solving Assignment-type Problems with Neighborhood Search Techniques* - Operations Research, Vol. 44, No. 2, Março-Abril 1996.

[Fink et al. 1998a] Fink, A.; Voss, S. e Woodruff, D. L. - *Building Reusable Software Components for Heuristic Search* - Tutorial, INFORMS/CORS Spring Meeting, Montreal, Abril, 1998.

[Fink et al. 1998b] Fink, A.; Voss, S. e Woodruff, D. L. - *Building Reusable Software Components for Heuristic Search* - Resumo Alongado, OR98, Zurique, 1998.

[Fink, Voss 1999] Fink, A. e Voss, S. - *Effects of Candidate List Strategies and Neighborhood Depth Variations on the Effectiveness and Efficiency of Meta-Heuristics* - Resumo alongado, Third Metaheuristics International Conference, Angra dos Reis, Brasil, Julho 19-23, 1999.

[Finkel, Bentley 1974] Finkel, R. A. e Bentley, J. L. - *Quad Trees, a Data Structure for Retrieval on Composite Keys* - Acta Informatica, 4, pp. 1-9, 1974.

[Fisher 1976] Fisher, M. L. - *A dual algorithm for the one-machine scheduling problem* - Mathematical Programming, 11, pp. 229-251, 1976.

[Fonseca, Fleming 1993] Fonseca, C. M. e Fleming, P. J. - *Genetic Algorithms for Multiobjective Optimization - Formulation, Discussion and Generalization* - Forrest, S. (ed.), Proceedings of the Fifth International Conference on Genetic Algorithms, pp. 416-423, San Mateo, California, University of Illinois at Urbana-Champaign, Morgan Kaufmann Publishers, 1993.

[Fonseca, Fleming 1995] Fonseca, C. M. e Fleming, P. J. - *An Overview of Evolutionary Algorithms in Multiobjective Optimization* - Evolutionary Computation, 3(1), pp. 1-16, Spring 1995.

[Fontoura et al. 1999] Fontoura, M.; Pree, W. e Rumpe, B. - *UML-FW - A Modeling Language for Object-Oriented Frameworks* - Technical Report 613-99, Princeton University, 1999.

[Fontoura et al. 2000] Fontoura, M.; Lucena, C. J.; Andreatta, A.; Carvalho, S. E. e Ribeiro, C. C. - *Using UML-FW to Enhance Framework Development: a Case Study in the Local Search Heuristics Domain* - 2000.

[Fortemps et al. 1994] Fortemps, P.; Teghem, J. e Ulungu, B. - *Heuristics for Multiobjective Combinatorial Optimization by Simulated Annealing* - XI-th International Conference on Multiple Criteria Decision Making, Coimbra, Portugal, 1-6.08.1994.

[Fourman 1985] Fourman, M . P. - *Compation of Symbolic Layout Using Genetic Algorithms* - Genetic Algorithms and Their Applications: Proceedings of the First International Conference on Genetic Algorithms, pp. 141-153, Lawrence Erlbaum, 1985.

[França et al. 1999] França, P.; Mendes, A. e Moscato, P. - *Memetic algorithms to minimize tardiness on a single machine with sequence-dependent setup times* - DSI'99 - 5th Intl. Conf. of the Decision Sciences Institute, Athens, Greece, 1999.

[French 1982] French, S. - *Sequencing and Scheduling: An Introduction to the Mathematics of the Job Shop* - Wiley, New York, 1982.

[Friesz et al. 1993] Friesz, T. L.; Anandalingam, G.; Mehta, N. J.; Nam, K.;Shah, S. J. e Tobin, R. L. - *The Multiobjective Equilibrium Netowrk Design Problem Revisited: A Simulated Annealing Approach* - European Journal of Operational Research, Vol. 65, pp. 44-57, 1993.

[Gamma et al. 1995] Gamma, E.; Helm, R.; Johnson, R. e Vlissides, J. - *Design Patterns: Elements of Reusable Object-Oriented Software* - Addison Wesley Professional Computing Series, Addison Wesley Longman, 1995.

[Gandibleux et al. 1996] Gandibleux, X.; Mezdaoui, N. e Fréville, A. - *A Tabu Search Procedure to Solve Multiobjective Combinatorial Optimization Problems* - Caballero, R. e Steuer, R. (eds.), Proceedings Volume of MOPGP '96, Springer-Verlag, 1996.

[Gervet 1998] Gervet, C. - *Large Combinatorial Optimization Problems: a Methodology for Hybrid Models and Solutions* - Journées Francophones de Programmation en Logique et par Constraintes, 1998.

[Glover 1963] Glover, Fred - *Parametric Combinations of Local Job Shop Rules* - ONR Research Memorandum no. 117, GSIA, Carnegie Mellon University, Pittsburgh, PA, Chapter IV, 1963.

[Glover 1986] Glover, Fred - *Future Paths for Integer Programming and Links to Artificial Intelligence* - Computers and Operations Research, Vol. 13, No. 5, pp. 533-549, 1986.

[Glover 1989] Glover, Fred - *Tabu Search - Part I* - ORSA Journal on Computing, Vol. 1, No. 3, pp. 190-206, 1989.

[Glover 1990] Glover, Fred - *Tabu Search - Part II* - ORSA Journal on Computing, Vol. 2, No. 1, pp. 4-32, 1990.

[Glover 1995a] Glover, Fred - *Scatter Search and Star-paths: Beyond the Genetic Metaphor* - OR Spectrum, Vol. 17, pp. 125-137, 1995.

[Glover 1995b] Glover, Fred - *Tabu Thresholding. Improved Search by Non-monotonic Trajectories* - ORSA Journal on Computing, Vol. 1, pp. 426, 1995.

[Glover, Laguna 1997] Glover, Fred e Laguna, M. - *Tabu Search* - Kluwer Academic Publishers, 1997.

[Glover 1998a] Glover, Fred - *Genetic Algorithms, Evolutionary Algorithms and Scatter Search: Changing Tides and Untapped Potentials* - INFORMS Computer Science Technical Section Newsletter, Vol. 19, No.1, 1998.

[Glover 1998b] Glover, Fred - *A Template for Scatter Search and Path Relinking* - Hao, J.-K.; Lutton, E.; Ronald, E.; Schoenauer, M. e Snvers, D. (eds.), Artificial Evolution, Lecture Notes in Computer Science 1363, pp. 13-54, Springer Verlag, 1998.

[Glover et al. 2000] Glover, Fred; Laguna, M. e Martí, R. - *Scatter Search* - Ghosh, A. e Tsutsui, S. (eds.), Theory and Application of Evolutionary Computation: Recent Trends, Springer Verlag (em publicação), London, 2000.

[Goldberg, Richardson 1987] Goldberg, D. E. e Richardson, J. - *Genetic Algorithm with Sharing for Multimodal Function Optimization* - Grefenstette, J. J. (ed.), Genetic Algorithms and Their Applications: Proceedings of the Second International Conference on Genetic Algorithms, pp. 41-49, Lawrence Erlbaum, 1987.

[Goldberg 1989] Goldberg, D. E. - *Genetic Algorithms in Search, Optimization and Machine Learning* - Addison-Wesley, 1989.

[Graccho, Porto 1999] Graccho, Márcio e Porto, Stella C. S. - *TabOOBuilder: An Object-Oriented Framework for Building Tabu Search Applications* - Resumo alongado, Third Metaheuristics International Conference, Angra dos Reis, Brasil, Julho 19-23, 1999.

[Graham et al. 1979] Graham, R. L.; Lawler, E. L.; Lenstra, J. K. e Rinnooy Kan, A. H. G. - *Optimization and approximation in deterministic sequencing and scheduling* - Annals of Discrete Mathematics, 5, pp. 287-326, 1979.

[Gupta, Kyparisis 1987] Gupta, S. K. e Kyparisis, J. - *Single Machine Scheduling Research* - Omega, 15, pp. 207-227, 1987.

[Gupta et al. 1993] Gupta, M .C.; Gupta, Y. P. e Kumar, A. - *Minimizing flow time variance in a single machine system using genetic algorithms* - European Journal of Operational Research, 70, pp. 289-303, 1993.

[Gutjahr 1998] Gutjahr, W. J. - *A Generalized Ant System and its Convergence* - Technical Report 98-10, Department of Statistics, Operations Research and Computer Science, University of Vienna, Austria, 1998.

[Habenicht 1982] Habenicht, W. - *Quad Trees, a Data Structure for Discrete Vector Optimization Problems* - Lecture Notes in Economics and Mathematical Systems, 209, pp. 136-145, 1982.

[Hajela, Lin 1992] Hajela, P. e Lin, C. Y. - *Genetic Search Strategies in Multicriterion Optuimal Design* - Structural Optimization, 4, pp. 99-107, 1992.

[Hansen 1986] Hansen, P. - *The Steepest Descent Mildest Descent Heuristic for Combinatorial Programming* - Apresentação, Numerical Methods in Combinatorial Optimization, Capri, Itália, 1986.

[Hansen 1997] Hansen, Michael P. - *Tabu Search for Multiobjective Optimization: MOTS* - The 13th International Conference on Multiple Criteria Decision Making, University of Cape Town, 6-10 January, 1997.

[Hansen 1997b] Hansen, Michael P. - *Experiments on the Usage of Hashing Vectors in Multiobjective Tabu Search* - Conference Paper, NOAS '97, Copenhagen, Denmmark, August 15-16, 1997.

[Hansen 1998] Hansen, Michael P. - *Metaheuristics for Multiple Objective Combinatorial Optimization* - Ph.D. Dissertation, Institute of Mathematical Modelling, Dissertation No. 45, Technical University of Denmark, 1998.

[Hansen, Mladenovic 1999] Hansen, P. e Mladenovic, N. - *Variable Neighborhood Search: Methods and Recent Applications* - Resumo alongado, Third Metaheuristics International Conference, Angra dos Reis, Brasil, Julho 19-23, 1999.

[Holland 1975] Holland, J. H. - *Adaptation in Natural and Artificial Systems* - The University of Michigan Press, Ann Arbor, Michigan, USA, 1975.

[Hoogeveen et al. 1997] Hoogeveen, J. A.; Lenstra, J. K. e van de Velde, S. L. - *Sequencing and Scheduling: an annotated bibliography* - Memorandum COSOR 97-02, Eindhoven University of Technology, 1997.

[Hopfield, Tank 1985] Hopfield, J. J. e Tank, D. W. - *Neural Computations of Decisions in Optimization Problems* - Biological Cybernetics, Vol. 52, pp. 141, 1985.

[Horn, Nafpliotis 1993] Horn, J. e Nafpliotis, N. - *Multiobjective Optimization Using the Niched Pareto Genetic Algorithm* - Technical Report IlliGAl Report 93005, University of Illinois at Urbana-Champaign, Urbana, Illinois, USA, 1993.

[Hosenback et al. 1999] Hosenback, J. E.; Russell, R. M.; Markland, R. E. e Philipoom, P. R. - *An improved heuristic for the single-machine, weighted tardiness problem* - Omega, 27, pp. 485-495, 1999.

[Hurink, Keuchel 2001] Hurink, J. e Keuchel, J. - *Local search algorithms for a single-machine scheduling problem with positive and negative time-lags* - Discrete Applied Mathematics, 112, pp. 179-197, 2001.

[IC-Parc 1998] IC-Parc, Imperial College - *ECLIPSE Version 3.7.0, User Manual* - 1998.

[ILOG 1997] ILOG, Inc. - *ILOG Solver, User Manual* - 1997.

[James, 1997] James, R. J. W. - *Using tabu search to solver the common due date early/tardy machine scheduling problem* - Computers and Operations Research, Vol. 24, No. 3, pp. 199-208, 1997.

[James, Buchannan 1997] James, R. J. W. e Buchannan, J. T. - *A neighbourhood scheme with a compressed solution space for the early/tardy scheduling problem* - European Journal of Operational Research, 102, pp. 513-527, 1997.

[James, Buchannan 1998] James, R. J. W. e Buchannan, J. T. - *Performance enhancements to tabu search for the early/tardy scheduling problem* - European Journal of Operational Research, 106, pp. 254-265, 1998.

[Johnson et al. 1989] Johnson, D. S.; Aragon, C. R.; McGeoch, L. A. e Schevon, C. - *Optimization by simulated annealing: an experimental evaluation: part I graph partitioning* - Operations Research, 37, pp. 865-892, 1989.

[Johnson, Foote 1998] Johnson, R. E. e Foote, B. - *Designing Reusable Classes* - Journal of Object-Oriented Programming, Vol. 1, No. 2, June 1988.

[Johnson 1992] Johnson, R. E. - *Documenting Framework Using Patterns* - OOPSLA '92 Proceedings, Vancouver, Outubro 1992.

[Johnson 1993] Johnson, R. E. - *How To Design Frameworks* - Tutorial notes, The 8th Conference on Object-Oriented Programming Systems, Languages and Applications, Washington, 1993.

[Keeney, Raiffa 1976] Keeney, R. L. e Raiffa, H. - *Decisions with Multiple Objectives: Preferences and Value Tradeoffs* - John Wiley & Sons, New York, 1976.

[Kirkpatrick et al. 1983] Kirkpatrick, S.; Gelatt, C. D. e Vecchi, M. P. - *Optimization by Simulated Annealing* - Science, Vol. 220, pp. 671-680, 1983.

[Kursawe 1991] Kursawe, F. - *A Variant of Evolution Strategies for Vector Optimization* - Schwefel, H. P. e Männer, R. (eds.), Parallel Problema Solving from Nature, 1st Workshop, PPSN I, Lecture Notes in Computer Science, Vol. 496, pp. 193-197, Berlin, Germany, Springer-Verlag, 1991.

[Laguna et al. 1991] Laguna, M.; Barnes, J. W. e Glover, F. W. - *Tabu search methods for a single machine scheduling program* - Journal of Intelligent Manufacturing, 2, pp. 63-74, 1991.

[Laguna, Glover 1993] Laguna, M. e Glover, F. W. - *Integrating target analysis and tabu search for improved scheduling systems* - Expert Systems Appl., 6, pp. 287-297, 1993.

[Lawler 1977] Lawler, E. L. - *A "pseudopolinomial" algorythm for sequencing jobs to minimize total tardiness* - Annals of Discrete Mathematics, 1, pp. 331-342, 1977.

[Lawler 1983] Lawler, E. L. - *Recent Results in the Theory of Machine Scheduling* - Mathematical Programming: The State of the Art, Bachem, A. et al. (eds.), pp. 203-234, Springer Verlag, 1983.

[Lawler et al. 1993] Lawler, E. L.; Lenstra, J. K.; Rinnooy Kan, A. H. G. e Shmoys, D. B. - *Sequencing and Scheduling: Algorithms and Complexity* - Logistics of Production and Inventory, Handbooks in Operations Research and Management Science 4, North-Holland, 1993.

[Lenstra et al. 1977] Lenstra, J. K.; Rinnooy Kan, A. H. G. e Brucker, P. - *Complexity of machine scheduling problems* - Annals of Discrete Mathematics, 1, pp. 343-362, 1977.

[Lenstra et al. 1982] Lenstra, J. K.; Rinnooy Kan, A. H. G. e Van Emde Boas, P. - *An Appraisal of Computational Complexity for Operations Research* - European Journal of Operational Research, Vol. 11, 1982.

[Lis, Eiben 1996] Lis, J. e Eiben, A. E. - *A Multi-sexual Genetic Algorithm for Multi-objective Optimization* - Fukuda, T. e Furuhashi, T. (eds.) Proceedings of the 1996 International Conference on Evolutionary Computation, pp. 59-64, Nagoya, Japan, IEEE, 1996.

[Liu, Tang 1999] Liu, J. e Tang, L. - *A modified genetic algorithm for single machine scheduling* - Computers & Indutrial Engineering, 37, pp. 43-46, 1999.

[Mackworth 1977] Mackworth, A. K. - *Consistency in Networks of Relations* - Artificial Intelligence, 1977.

[Madureira 1995] Madureira, A. - *Aplicação de Meta-heurísticas em Problemas de Sequenciamento* - Tese de Mestrado, Faculdade de Engenharia da Universidade do Porto, 1995.

[Madureira, Sousa 1996] Madureira, A. M. e Sousa, J. P. - *Aplicação de Meta-heurísticas a Problemas de Escalonamento de uma Única Máquina* - Investigação Operacional, Vol. 16, pp. 115-133, Dezembro 1996.

[Matos, Ponce de Leão 1995] Matos, M. A. e Ponce de Leão, M . T. - *Electric Distribution Systems Planning with Fuzzy Loads* - International Transactions of Operational Research, Vol. 2, No. 3, pp. 287-296, 1995.

[Mattsson 1996] Mattsson, M. - *Object-Oriented Frameworks - A Survey of Methodological Issues* - LU-CS-TR 96-167, Department of Computer Science, Lund University, Sweden , 1996.

[Mehlhorn, Näher 1995] Mehlhorn, K. e Näher, S. - *LEDA: A platform for combinatorial and geometric computing* - Communications of the ACM, 38: 96-102, 1995.

[Mendes, Aguilera 1998] Mendes, A. S. e Aguilera, L. M. - *A Hopfield neural network approach to the single machine scheduling problem* - Pre-prints of IFAC-INCOM'98 - Information Control in Manufacturing, Nancy-Metz, France, June, 1998.

[MIC 1999] Extended Abstracts of the Third Metaheuristics International Conference, Angra dos Reis, Brasil, Julho 19-23, 1999.

[Microsoft 1998] Microsoft Corporation - *MSDN Library Visual Studio 6.0 release* - Microsoft Corporation, 1998.

[Michel, van Hentenryck 1998] Michel, L. e van Hentenryck, P. - *Localizer* - Technical Report CS-98-02, Brown University, Computer Science Department, 1998.

[Mladenovic, Hansen 1997] Mladenovic, N. e Hansen, P. - *Variable Neighborhood Search* - Computers and Operations Research, Vol. 24, No. 11, pp. 1097-1100, 1997.

[Musser, Saini 1996] Musser, D. R. e Saini, A. - *STL Tutorial and Reference Guide* - Addison-Wesley, Reading, 1996.

[Nagar et al. 1995] Nagar, A.; Haddock, J. e Heradu, S. - *Multiple and Bicriteria Scheduling: A Literature Survey* - European Journal of Operational Research, Vol. 81, pp.88-104.

[Nievergelt 1994] Nievergelt, J. - *Complexity, Algorithms, Programs, Systems: The Shifting Focus* - Journal of Symbolic Computation, 17, pp. 297-310, 1994.

[OMG 1999] Object Mangement Group - *OMG Unified Modeling Language Specification, Version 1.3* - Object Management Group, 1999.

[Osyczka, Kundu 1995] Osycka, A. e Kundu, S. - *A New Method to Solve Generalized Multicriteria Optimization Problems Using the Simple Genetic Algorithm* - Structural Optimization, 10, pp. 94-99, 1995.

[Papadimitriou, Steiglitz 1982] Papadimitriou, C. H. e Steiglitz, K. - *Combinatorial Optimization* - Wiley, New York, 1982

[Périaux et al. 1997] Périaux, J.; Sefrioui, M. e Mantel, B. - *GA Multiple Objective Optimization Strategies for Electromagnetic Backscattering* - Quagliarella, D.; Périaux, J.; Poloni, C. e Winter, G. (eds.), Genetic Algorithms and Evolution Strategies in Engineering and Computer Science - Recent Advances and Industrial Applications, cap. 11, pp. 225-243, John Wiley & Sons, West Sussex, England, 1997.

[Pinedo, Chao 1999] Pinedo, M. e Chao, X. - *Operations Scheduling with Applications in Manufacturin and Services* - McGraw-Hill International Editions, Computer Science Series, 1999.

[Pirlot 1996] Pirlot, Marc - *General Local Search Methods* - European Journal of Operational Research, Vol. 92, pp. 493-511, 1996.

[Potts, van Wassenhove 1985] Potts, C. N. e van Wassenhove, L. N. - *A branch and bound algorithm for the total weighted tardiness problem* - Operations Research, 33, pp. 363-377, 1985.

[Potts, van Wassenhove 1987] Potts, C. N. e van Wassenhove, L. N. - *Dynamic programming and decomposition approaches for the single machine total tardiness problem* - Operations Research, 33, pp. 363-377, 1987.

[Potts, van Wassenhove 1991] Potts, C. N. e van Wassenhove, L. N. - *Single Machine Tardiness Sequencing Heuristics* - IIE Transactions, 23, pp. 346-354, 1991.

[Pressman 1992] Pressman, R. - *Software Engineering: A Practitioner's Approach* - 3rd Ed., McGraw-Hill, 1992.

[Puget 1994] Puget, J. - F. - *A C++ Implementation of CLP* - Proceedings of SPICIS 94: The Second Singapore International Conference on Intelligent Systems, 1994.

[Puget 1995] Puget, J. - F. e Leconte, M. - *Beyond the glass box: constraints as objects* - International Logic Programming Symposium, MIT Press, 1995.

[Rachamadugu 1987] Rachamadugu, R. M. - *A note on weighted tardiness problem* - Operations Research, 35, pp. 405-414, 1987.

[Rangaswamy et al. 1998] Rangaswamy, B.; Jain, A. S. e Glover, Fred - *Tabu Search Candidate List Strategies in Scheduling* - Woodruff, D. L. (ed.), 6th INFORMS Advances in Computational and Stochastic optimization, Logic Programming and Heuristic Search: Interfaces in Computer Science and Operations Research Conference, Monterrey Bay, California, Jan 1998, Kluwer Academic Publishers, 1998.

[Rinnooy Kan et al. 1975] Rinnooy Kan, A. H. G.; Lageweb, B. J. e Lenstra, J. K. - *Minimizing total costs in one-machine scheduling* - Operations Research, 23, pp. 908-927, 1975.

[Ritzel et al. 1994] Ritzel, B. J.; Eheart, J. W. e Ranjithan, S. - *Using Genetic Algorithms to Solve a Multiple Objective Groundwater Pollution Containment Problem* - Water Resources Research, 30(5), pp. 1589-1603, 1994.

[Rosing, ReVelle 1997] Rosing, K. E. e ReVelle, C. S. - *Heuristic Concentration : Two Stage Solution Construction* - European Journal of Operational Research, Vol. 97, pp. 75-86, 1997.

[Russo 1991] Russo, V. - *An Object-Oriented Oeprating System* - Ph.D. Thesis, University of Illinois, Urbana-Champaign, 1991.

[Schaerf et al. 1999] Schaerf, A.; Lenzerini, M. e Cadoli, M. - *Local++: A C++ Framework for Local Search Algorithms* - Technical Report 11-99, Dipartimento di Informatica e Sistemistica, Universita di Roma, La Sapienza, May 1999.

[Schaffer 1985] Schaffer, J. D. - *Multiple Objective Optimization with Vector Evaluated Genetic Algorithms* - Genetic Algorithms and Their Applications: Proceedings of the First International Conference on Genetic Algorithms, pp. 93-100, Lawrence Erlbaum, 1985.

[Serafini 1987] Serafini, P. - *Some Considerations About Computational Complexity for Multiobjective Combinatorial Problems* - LNEMS, Vol. 294, Springer Verlag, pp. 222-232, 1987.

[Serafini 1992] Serafini, P. - *Simulated Annealing for Multiple Objective Optimization Problems* - Tzeng, G. H.; Wang, H. F.; Wen, V. P. e Yu, P. L. (eds.), Multiple Criteria Decision Making - Expand and Enrich the Domains of Thinking and Application, Springer Berlag, pp. 283-292, 1992.

[Silver et al. 1980] Silver, E. A.; Vidal. R. V. e de Werra, D. - *A Tutorial on Heuristic Methods* - European Journal of Operational Research, Vol. 5, 1980.

[Slowinski 1981] Slowinski, R. - *Multiobjective Network Scheduling with Efficient Use of Renewable and Non Renewable Resources* - European Journal of Operational Research, 7, pp. 265-273, 1981.

[Slowinski 1989] Slowinski, R. - *Multiobjective Project Scheduling under Multiple-category Resource Constraint* - Slowinski, R. e Weglarz, J. (eds.), Advances in Project Scheduling, Elsevier, Asmsterdam, pp. 151-167, 1989.

[Sousa 1989] Sousa, J. P. - *Time indexed formulations of non-preemptive single-machine scheduling problems* - Ph. D. Thesis, Faculté des Sciences Apliquées, Université Catholique de Louvain, Belgium, 1989.

[Srinivas, Deb 1993] Srinivas, N. e Deb, K. - *Multiobjective Optimization Using Nondominated Sorting in Genetic Algorithms* - Technical Report, Department of Mechanical Engineering, Indian Institute of Technology, Kanput, India, 1993.

[Steuer 1986] Steuer, Ralph E. - *Multiple Criteria Optimization: Theory, Computation and Application* - John Wiley & Sons, Inc., 1986.

[Syswerda, Palmucci 1991] Syswerda, G. e Palmucci, J. - *The Application of Genetic Algorithms to Resource Scheduling* - Belew, R. K. e Booker, L. B. (eds.), Proceedings of the Fourth International Conference on Genetic Algorithms, pp. 502-508, San Mateo, California, Morgan Kaufman, 1991.

[Talbi 1998] Talbi, E. - G. - *A Taxonomy of Hybrid Metaheuristics* - Laboratoire d'Informatique Fondamentale de Lille, Université de Lille 1, TR-183, Mai 1998.

[Taligent 1994] Taligent Inc. - *Building Object-Oriented Frameworks* - A Taligen White Paper, 1994.

[Tan, Narasimhan 1997] Tan, K. C. e Narasimhan, R. - *Minimizing tardiness on a single processor with sequence-dependent setup time: a simulated annealing approach* - OMEGA, 25, No. 6, pp. 619-634, 1997.

[Thienel 1995] Thienel, Stefan - *ABACUS A Branch-And-CUt System* - Inaugural-Dissertation zur Erlangung des Doktorgrades der Mathematisch-Naturwissenschaftlischen Fakultät der Universität zu Köln, 1995.

[Toulouse et al. 1997] Toulouse, M.; Crainic, T. G. e Sanso, B. - *Systemic Behavior of Cooperative Search Algorithms* - Technical Report, Centre de Recherche sur Les Transports, University of Montreal, Canada, 1997.

[Tsumakitani, Evans 1998] Tsumakitani, S. e Evans, J. R. - *An Empirical Study of a New Metaheuristic for the Traveling Salesman Problem* - European Journal of Operational Research, Vol. 104, pp. 113-128, 1998.

[Ulungu, Teghem 1991] Ulungu, E. L. e Teghem, J. - *The Multi-objective Shortest Path Problem: A Survey* - Cerny, M.; Gluckaufova, D. e Loula, D. (eds.), Proceedings of the International Workshop "Multicriteria Decision Making Methods - Algorithms - Applications", Liblice, Czechoslovakia, pp. 176-188, 1991.

[Ulungu, Teghem 1994] Ulungu, E. L. e Teghem, J. - *Multi-objective Combinatorial Optimization Problems: a Survey* - Journal of Multiple-criteria Decision Analysis, Vol. 3, pp. 83-104, 1994.

[Ulungu et al. 1997] Ulungu, E. L.; Teghem, J.; Fortemps, Ph. e Tuyttens, D. - *MOSA Method: a Tool for Solving Multi-Objective Combinatorial Optimization Problems* - Laboratory of Mathematics & Operational Research, Faculté Polytechnique de Mons, Belgium, 1997.

[Vaessens et al. 1995] Vaessens, R. J. M.; Aarts, E. H. L. e Lenstra, J. K. - *A Local Search Template* - Technical Report COSOR 92-11 (revised version), Eindhoven University of Technology, Enindhoven, NL, 1995

[Valenzuela-Rendón, Uresti-Charre 1997] Valenzuela-Rendón, M. e Uresti-Charre, E. - *A Non-generational Genetic Algorithm for Multiobjective Optimization* - Bäck, T. (ed.), Proceedings of the Seventh International Conference on Genetic Algorithms, pp. 658-665, San Mateo, California, Michigan State University, Morgan Kaufmann Publishers, 1997.

[Van Hentenryck 1989] Van Hentenryck, P. - *A Logic Language for Combinatorial Optimization* - Annals of Operations Research, 21, pp. 247-273, 1989.

[Vepsalainen, Morton 1987] Vepsalainen, A. e Morton, T. E. - *Priority Rules and Lead Time Estimation for Job Shop Scheduling with Weighted Tardiness Costs* - Management Science, 33, pp.1036-1047, 1987.

[Viana 1997] Viana, A. - *Aplicação de Meta-heurísticas Multiobjectivo ao Problema de Sequenciamento de Actividades com Restrições de Recursos* - Tese de Mestrado, Faculdade de Engenharia da Universidade do Porto, 1997.

[Vidal 1993] Vidal, R. V. V. (ed.) - *Applied Simulated Annealing* - LNEMS Vol. 396, Springer-Verlag, 1993.

[Vincke 1989] Vincke, P. - *Decision-Aid* - John Wiley & Sons, 1989.

[Vincke 1995] Vincke, P. - *Analysis of MCDA in Europe* - European Journal of Operational Research, 25, pp. 160-168, 1995.

[Wienke et al. 1992] Wienke, P . B.; Lucasius, C. e Kateman, G. - *Multicriteria Target Optimization of Analytical Procedure Using a Genetic Algorithm* - Analytical Chimica Acta, 265(2), pp. 211-225, 1992.

[White 1990] White, D. J. - *A Bibliography on the Applications of Mathematical Programming Multiple-objective Methods* - Journal of the Operational Research Society, 41, pp. 669-691, 1990.

[Wilf 1994] Wilf, Herbert S. - *Algorithms and Complexity* - Internet Edition, Summer, 1994. Disponível em http://www.math.upenn.edu.

[Wilson, MacLeod 1993] Wilson, P. B. e Macleod, M. D. - *Low Implementation Cost IIR Digital Filter Design Using Genetic Algorithms* - IEE/IEEE Workshop on Natural Algorithms in Signal Processing, pp. 4/1-4/8, Chelmsford, U. K., 1993.

[Wilson, Wilson 1993] Wilson, D. A. e Wilson, S. D. - *Writing Frameworks - Capturing Your Expertise About a Problem Domain* - Tutorial notes, The 8th Conference on Object-Oriented Programming Systems, Languages and Applications, Washington, 1993.

[Woodruff 1997] Woodruff, D. L. - *A Class Library for Heuristic Search Optimization* - INFORMS Computer Science Technical Section, 18(2), pp. 1-5, 1997.

[Yagiura, Ibaraki 1995] Yagiura, M. e Ibaraki, T. - *Genetic and local search algorithms as robust and simple optimization tools* - Comunicação, First Metaheuristics International Conference, 1995.

[Yannakakis 1990] Yannakakis, M. - *The Analysis of Local Search Problems and Their Heuristics* - Proceedings 7th Annual Symposium on Theoretical Aspects of Computer Science (STACS 90), Lecture Notes in Computer Science, 415, Springer, Berlin, pp. 298-311, 1990.

[Zanakis, Evans 1981] Zanakis, S. H. e Evans, J. R. - *Heuristic 'Optimization': Why, When and How to Use It* - Interfaces, Vol. 11, No. 5, Outubro 1981.

[Zimmer 1994] Zimmer, W. - *Relationships between Design Patterns* - Proceedings of the First Conference on Pattern Languages and Programming, Addison-Wesley, 1994.