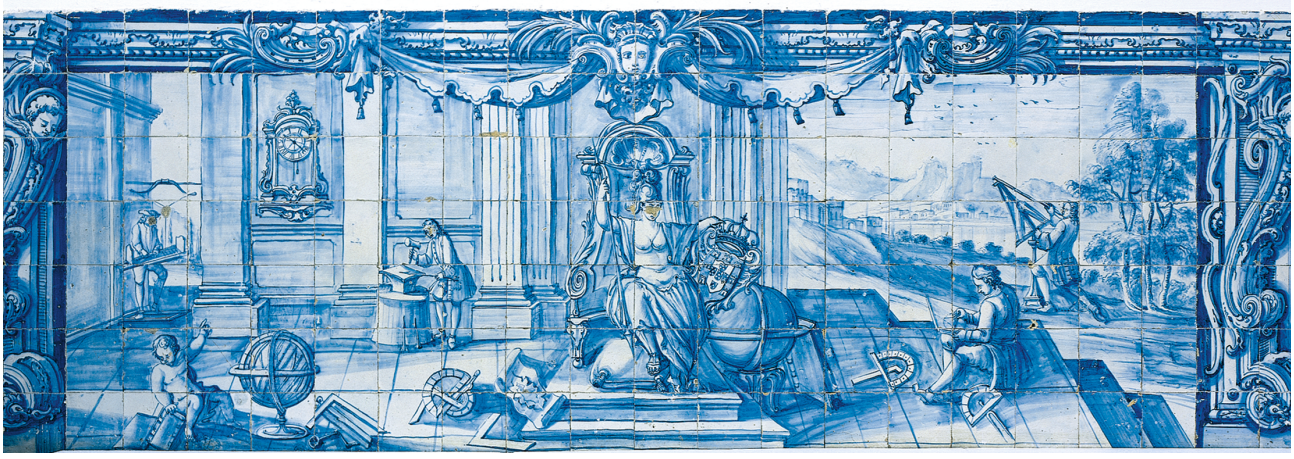
Optimização de Métodos de Núcleo Utilizando Algoritmos de Enxame

Arlindo Ferreira da Silva

Tese apresentada à Universidade de Évora
para obtenção do Grau de Doutor em Informática

ORIENTADORA: *Teresa Cristina de Freitas Gonçalves*

ÉVORA, Julho de 2014



INSTITUTO DE INVESTIGAÇÃO E FORMAÇÃO AVANÇADA

Optimização de Métodos de Núcleo Utilizando Algoritmos de Enxame

Tese apresentada à
Universidade de Évora
para obtenção do Grau de Doutor
em Informática

por

Arlindo Ferreira da Silva

Orientadora: Professora Doutora Teresa Cristina de Freitas Gonçalves

Universidade de Évora
Escola de Ciências e Tecnologia
Departamento de Informática

Julho de 2014

Optimização de Métodos de Núcleo Utilizando Algoritmos de Exame

Para a Paula, a Beatriz e a Catarina - as partículas do meu exame.

Agradecimentos

Gostaria de agradecer, em primeiro lugar, à minha orientadora, Professora Teresa Gonçalves, pela sua disponibilidade e incentivo num percurso longo e sujeito a restrições difíceis de conciliar. A sua compreensão e apoio foram essenciais para que o fim desse percurso fosse atingido.

Em relação ao outro extremo do caminho, não posso deixar de agradecer ao Professor Ernesto Costa, a quem devo a introdução à área da computação evolucionária e muitos anos de orientação bem-humorada.

Ao corpo docente do Departamento de Informática da Universidade de Évora, agradeço o acolhimento afável e as contribuições e comentários oferecidos durante as apresentações públicas nas diversas fases deste trabalho.

Já em relação à minha Instituição de origem, quero agradecer o companheirismo dos colegas da Unidade Técnico-Científica de Informática, e, em particular, aos restantes elementos da área científica de Programação, Algoritmos e Desenvolvimento de Software.

Ainda em relação ao Instituto Politécnico de Castelo Branco, quero vincar o meu agradecimento pelo apoio aos meus trabalhos de doutoramento, no âmbito do programa de formação avançada, o qual foi sendo mantido dentro das possibilidades da Instituição e em condições progressivamente mais difíceis.

Também à Direcção da Escola Superior de Tecnologia devo os meus agradecimentos por, dentro do possível, ter permitido a conciliação das minhas actividades lectivas com a investigação necessária à realização deste trabalho.

Arlindo Silva

Castelo Branco, Setembro de 2013

Optimização de Métodos de Núcleo Utilizando Algoritmos de Enxame

Resumo

Os métodos de núcleo, que incluem as máquinas de vectores de suporte, têm obtido sucesso assinalável em inúmeras áreas de aplicação. Este sucesso está, no entanto, dependente do núcleo e parâmetros definidos para cada problema. Estas questões têm sido abordadas utilizando diversas técnicas analíticas, algébricas, heurísticas e, mais recentemente, vários métodos evolucionários. Nesta dissertação apresentamos uma abordagem baseada em inteligência de enxame à optimização das máquinas de vectores de suporte. As contribuições deste trabalho incluem a análise de anteriores abordagens evolucionárias; a proposta de um algoritmo de enxame para treino de máquinas de vectores de suporte, superior à melhor abordagem evolucionária conhecida e aos métodos clássicos quando são utilizados núcleos indefinidos; e a apresentação de um algoritmo de enxame capaz de optimizar simultaneamente vários aspectos da abordagem, reduzindo substancialmente a intervenção do utilizador. Adicionalmente, desenvolvemos novos algoritmos e metodologias de utilidade genérica na área da inteligência de enxame.

Optimizing Kernel Methods Using Swarm Intelligence Algorithms

Abstract

Kernel methods, which include support vector machines, have achieved remarkable success in many application areas. This success is, however, dependent on the kernel and parameters defined for each problem. These issues have been addressed using various analytical, algebraic and heuristic techniques. More recently, various evolutionary methods have also been used. In this thesis we present an approach based on swarm intelligence for the optimization of support vector machines. The contributions of this work include the analysis of previous evolutionary approaches; the proposal of a swarm algorithm to train support vector machines that presents better performance than the best previously known evolutionary approach and can overcome the classical methods when indefinite kernels are used; and the presentation of a swarm algorithm able to simultaneously optimize several aspects of the approach, substantially reducing user intervention. Additionally, we developed new algorithms and methodologies of general utility in the area of swarm intelligence.

Índice

Lista de Tabelas

Lista de Figuras

Lista de Algoritmos

Lista de Acrónimos

ABO	Aprendizagem Baseada em Oposição
AG	Algoritmo Genético
AE	Algoritmo Evolucionário
EE	Estratégia Evolutiva
EE-AMC	Estratégia Evolutiva com Adaptação da Matriz de Co-variância
ED	Evolução Diferencial
EDPL	Evolução Diferencial com Procura Livre
FBR	Função de Base Radial
MVS	Máquina de Vectores de Suporte
OCF	Optimização por Colónias de Formigas
OEP	Optimização por Enxames de Partículas
OEPHPMA	OEP Híbrido com Procura Moderadamente Aleatória
OLEP	Optimização Linear por Enxames de Partículas
OPP	Optimizador Predador-Presa
OPPB	Optimizador Predador-Presa com Batedores
OPPL	Optimizador Predador-Presa com Procura Local
OPPO	Optimizador Predador-Presa com Procura por Oposição
OSM	Optimização Sequencial Mínima
PG	Programação Genética
PL	Procura Livre

PQ	Programação Quadrática
PSD	Positivo Semi-Definido
RSD	Resultante da Substituição da Distância

Capítulo 1

Introdução

Os métodos de núcleo são técnicas de análise de dados cuja estratégia base consiste em mapear os dados originais num espaço secundário de características, onde os padrões existentes possam ser descritos através de relações lineares [Shawe-Taylor and Cristianini, 2004]. Este processo é realizado de forma modular, já que cada um dos passos é implementado por componentes distintas.

A componente de mapeamento dos dados é definida implicitamente através de uma função de núcleo[1]. Esta depende não só dos dados específicos do problema, mas também do conhecimento acerca dos padrões que se esperam encontrar. Neste novo espaço de características são então procurados padrões lineares, através da utilização de algoritmos de aprendizagem de carácter geral e aplicáveis a diversos problemas [Hofmann et al, 2008].

Os representantes mais conhecidos destes métodos são as máquinas de vectores de suporte (MVS) [Cristianini and Scholkopf, 2002; Cristianini and Shawe-Taylor, 2000], utilizadas sobretudo como algoritmo de classificação. O assinalável sucesso com que estes métodos têm sido aplicados a uma diversidade de áreas resulta directamente das suas características específicas:

- eficiência computacional, já que requerem uma quantidade de recursos computacionais de ordem polinomial em relação ao número de exemplos, permitindo trabalhar com muitos dados de elevada dimensionalidade;
- robustez, já que possuem bases teóricas bem estabelecidas, originárias da aprendizagem estatística, que lhes conferem resistência ao sobre-ajustamento;
- generalidade de aplicação, já que a escolha adequada da função de núcleo permite

[1] Do inglês *kernel*.

aprender funções de decisão não lineares e trabalhar com dados não-vectoriais e heterogéneos.

Apesar do sucesso desta abordagem, a aplicação de MVS a um novo problema apresenta uma série de dificuldades e elementos de escolha dos quais a qualidade dos resultados finais depende. Entre os diversos elementos cuja definição é necessária temos:

- Selecção de atributos - Esta é uma questão comum a todos os algoritmos de aprendizagem, já que uma selecção prévia dos atributos ou características a considerar no processo de aprendizagem, eliminando atributos redundantes ou inúteis, permite poupanças óbvias do ponto de vista computacional, além de possibilitar modelos preditivos mais simples.
- Escolha da função de núcleo - A escolha de uma função de núcleo apropriada é uma questão complexa e determinante para o desempenho do algoritmo, já que o mapeamento do espaço dos dados para o espaço de classificação é implicitamente definido por esta função. A possibilidade do algoritmo de optimização encontrar um classificador com elevado desempenho preditivo depende desse mapeamento.
- Definição de uma nova função de núcleo - A necessidade de adequar a função de núcleo à especificidade do problema pode levar à necessidade de definir novas funções de núcleo. Esta tarefa pode ser complexa, já que as funções de núcleo devem ser positivas semi-definidas de maneira a que os algoritmos de treino possam garantidamente encontrar um óptimo global.
- Optimização dos parâmetros da função de núcleo - A generalidade das funções de núcleo mais utilizadas têm parâmetros que é necessário escolher cuidadosamente.
- Escolha do algoritmo de treino - O problema de optimização da MVS resultante da utilização de uma função de núcleo positiva semi-definida é um problema de programação quadrática, existindo várias implementações que permitem encontrar o máximo global. Se o núcleo não for positivo semi-definido (PSD), estes métodos não garantem que encontram um máximo ou mesmo que a sua execução termina.
- Optimização do parâmetro C - Na função a optimizar no processo de treino, o parâmetro C permite definir o equilíbrio entre o desempenho preditivo do modelo final e sua qualidade. O valor do parâmetro tem de ser escolhido de maneira a que o desempenho seja elevado, simultaneamente evitando o sobre-ajustamento do modelo.

Estas questões têm sido abordadas utilizando técnicas tanto analíticas e algébricas, como baseadas em mecanismos de procura heurística. Recentemente têm sido utilizados

métodos evolucionários, tais como algoritmos genéticos e programação genética. Estes algoritmos apresentam vantagens quando a procura e/ou optimização é feita em espaços complexos, não vectoriais, ou quando a função a optimizar é multi-modal, apresentando vários óptimos locais em alternativa a uma única solução.

1.1 Motivação e Objectivos

A motivação essencial para o trabalho descrito nesta dissertação decorre da constatação de que, para um utilizador não especialista, a definição de todos os aspectos necessários à aplicação de MVS a um determinado problema pode constituir uma tarefa complexa e até desencorajadora. Da análise da bibliografia que descreve a aplicação de computação evolucionária a alguns dos problemas atrás listados, podemos concluir que esta área apresenta abordagens interessantes a vários desses problemas, as quais nos parecem merecer ser aprofundadas, melhoradas e integradas numa solução que permita a um utilizador comum aplicar esta metodologia tão promissora a um problema específico, fornecendo pouco mais do que os dados de treino do problema.

Da mesma análise da bibliografia, que apresentamos num capítulo posterior, foi possível identificar quatro áreas essenciais na aplicação de máquinas de vectores de suporte a um problema concreto nas quais podemos agrupar as abordagens evolucionárias: selecção de atributos ou características, definição do núcleo a utilizar, optimização dos diversos parâmetros e o próprio algoritmo de treino da MVS. Destas, este último aspecto é o menos explorado e também um dos que maior interesse prático tem, já que a existência de algoritmos de treino evolucionários abre a porta à utilização de funções de núcleo úteis mas não garantidamente PSD, para as quais os métodos de treino tradicionais, baseados em programação quadrática, não garantem que a solução óptima, para o problema de optimização resultante, é encontrada [Haasdonk, 2005].

Este facto, além de também facilitar a aplicação do método, já que a demonstração da positividade do núcleo pode ser uma tarefa difícil, adiciona um segundo nível de motivação, pois o treino de MVS com núcleo não PSD constitui um problema interessante por diversos motivos: existem núcleos não PSD com interesse prático, como, por exemplo, o núcleo sigmóide [Lin and Lin, 2003]; alguns núcleos não PSD apresentam bons resultados empíricos, mesmo com os métodos de treino tradicionais [Haasdonk and Bahlmann, 2004; Lin and Lin, 2003]; vários métodos de aprendizagem de funções de núcleo (incluindo alguns de origem evolucionária) devolvem núcleos não garantidamente PSD [Ong et al, 2004]; pode ser utilizado em conjunção com formulações evolucionárias multi-objectivo das MVS, as quais permitem a optimização independente do erro e da complexidade do modelo, i.e., não é necessária a utilização do parâmetro C [Mierswa, 2007].

Para as restantes três áreas, embora existam abordagens interessantes em cada uma, em geral estas estão orientadas para um aspecto em particular, ou à combinação de dois aspectos (por exemplo selecção de atributos e optimização simultânea de alguns parâmetros), não existindo soluções evolucionárias integradas, as quais permitam definir o núcleo, escolher os parâmetros e optimizar o problema resultante utilizando um algoritmo de treino evolucionário. O desenvolvimento de uma abordagem deste tipo seria um contributo essencial para responder ao problema correspondente à nossa motivação principal. Uma abordagem integrada poderia devolver ao utilizador, para qualquer conjunto de dados fornecido, um núcleo adequado, possivelmente não PSD, um conjunto de parâmetros adequado e uma MVS treinada, caso tal seja necessário, por um optimizador evolucionário.

1.2 Abordagem Proposta

Neste trabalho propomos a utilização de algoritmos provenientes de uma área particular da computação evolucionária, a inteligência de enxame, para tentar responder às questões levantadas na secção anterior. A escolha destes algoritmos em particular está relacionada com o nosso conhecimento anterior desta sub-área da computação evolucionária, em especial dos algoritmos de optimização por enxames de partículas. As características destes algoritmos, incluindo a simplicidade conceptual, facilidade de implementação, custo computacional limitado (funciona bem com populações/enxames pequenos), rapidez de convergência, facilidade de hibridização com outras abordagens e de aplicação a domínios variados, tornam-os candidatos adequados aos nossos objectivos.

Uma das maiores dificuldades de aplicação de um algoritmo evolucionário a problemas de classificação, especialmente quando envolvem conjuntos de treino com um elevado número de exemplos e a avaliação inclui alguma forma de validação cruzada, tem a ver com a complexidade computacional de cada execução do algoritmo. O desenvolvimento de um algoritmo eficiente, necessitando de inúmeras execuções, comparações e testes com diferentes conjunto de dados de treino, torna-se assim difícil quando se trabalha directamente com os problemas de classificação.

Neste trabalho optámos por uma abordagem diferente. O desenvolvimento de um algoritmo de treino eficiente foi dividido em duas fases. Numa primeira fase, procurámos identificar as características básicas dos problemas de optimização correspondentes ao treino de MVS baseadas em núcleos não PSD. Posteriormente foram seleccionadas funções padrão da área da optimização numérica onde é possível observar o comportamento de um algoritmo de optimização em relação àquelas características. A utilização destas funções padrão na avaliação do algoritmo, em vez do uso directo dos problemas de

classificação, permitiu acelerar consideravelmente o trabalho experimental necessário ao seu desenvolvimento, incluindo a comparação com outros algoritmos correspondentes ao estado da arte da área.

Numa segunda fase, o algoritmo de optimização já desenvolvido foi integrado num algoritmo específico de treino de máquinas de vectores de suporte e testado exaustivamente utilizando um conjunto de problemas de classificação padrão, retirados da área da aprendizagem automática. Estes testes incluíram várias funções de núcleo correspondentes a diversas situações de interesse prático, de forma a ilustrar não só a viabilidade, mas também a utilidade do treino evolucionário de MVS. O desempenho da abordagem baseada em inteligência de enxame foi ainda comparado tanto com as abordagens clássicas mais populares, como com a melhor abordagem evolucionária conhecida.

Estabelecida a eficiência do novo algoritmo de treino, na última fase do trabalho foi realizada a sua integração numa abordagem de dois níveis, totalmente baseada em algoritmos de inteligência de enxame, à optimização dos diversos aspectos da utilização de uma MVS num novo problema de classificação. Também este novo algoritmo foi testado no conjunto de problemas de teste, de maneira a determinar a competitividade dos núcleos e parâmetros devolvidos, bem como o desempenho da própria metodologia de treino, quando comparados com uma abordagem clássica de aplicação de MVS a um novo problema.

1.3 Contributos Principais

O trabalho descrito nesta dissertação produziu contributos significativos em duas áreas distintas, facto que pode ser atribuído, pelo menos parcialmente, à forma como esse trabalho foi organizado. O desenvolvimento do algoritmo de enxame destinado ao treino de máquinas de vectores de suporte, ao ser feito de forma independente em relação aos problemas de classificação, permitiu desenvolver mecanismos, que, além de úteis a este problema em particular, resultaram num algoritmo de optimização bastante competitivo em problemas de optimização genéricos, quando comparado com versões estado da arte de algoritmos de optimização baseados em enxames. Entre os contributos inovadores deste novo algoritmo, que denominámos optimizador predador-presa com batedores, podemos salientar:

- A utilização de uma partícula específica, denominada predador, que, em conjunção com um mecanismo de perturbação predador-presa, permite, de forma adaptativa, melhor controlar o equilíbrio entre a exploração local e a exploração global do espaço de procura realizada pelo enxame durante o processo de

optimização.

- A sugestão de utilização de partículas batedoras, as quais usam regras de actualização específicas, distintas da utilizada pelas restantes partículas do enxame, para implementar comportamentos de exploração específicos. Estes permitem integrar no algoritmo, tanto mecanismos que melhoram o seu desempenho globalmente (e.g. procura local), como melhorias específicas resultantes da integração de conhecimento sobre o problema concreto a abordar.

- O sublinhar da heterogenia do enxame, um aspecto da inteligência de enxame que tem recebido atenção crescente, como mecanismo valioso no desenvolvimento de novas variantes dos optimizadores por enxames de partículas.

- A manutenção da integridade da metáfora de inteligência de enxame, utilizando mecanismos comummente associados a enxames reais, como a predação, a especialização de diferentes grupos ou castas num mesmo enxame ou a utilização de elementos batedores, como inspiração para novos mecanismos algorítmicos, por oposição às abordagens mais comuns baseadas na hibridização do algoritmo com outras abordagens (e.g., através da utilização de operadores de mutação).

O algoritmo desenvolvido foi integrado num algoritmo de treino de MVS e este foi posteriormente utilizado numa abordagem global, baseada em inteligência de enxame, à aplicação de máquinas de vectores de suporte a um novo problema. Este processo resultou em diversos contributos para a área da integração da computação evolucionária e MVS:

- Neste trabalho apresentamos o primeiro algoritmo de treino de MVS baseado em inteligência de enxame que é competitivo, tanto com as abordagens clássicas, como com as melhores abordagens evolucionárias, no treino de MVS com núcleos PSD. Embora os problemas de optimização resultantes, sendo unimodais, não sejam particularmente complexos, levavam mesmo assim abordagens anteriores encontradas na bibliografia (e parcialmente reproduzidas por nós) a estagnar prematuramente, produzindo classificadores com pior desempenho do que os resultantes de outras abordagens. O mesmo algoritmo foi aplicado com sucesso ao treino de MVS com diversos núcleos não PSD, no que constitui também a primeira utilização, de que temos conhecimento, de inteligência de enxame na abordagem a este problema.

- A comparação, em termos de desempenho nos problemas de optimização correspondente ao treino de MVS, com a melhor abordagem evolucionária conhecida, permite concluir que o algoritmo proposto é, das propostas baseadas em computação evolucionária, a mais eficiente no problema de optimização, i.e., a que

necessita de menos avaliações da função objectivo para produzir soluções de determinada qualidade.

- O estudo experimental realizado com núcleos não PSD é o mais extenso realizado até à data (na bibliografia da área apenas se encontram resultados utilizando abordagens evolucionárias para um núcleo), permitindo mais claramente inferir as vantagens do treino evolucionário de MVS baseadas em núcleos não PSD, quando comparado com os algoritmos baseados em programação quadrática.

- Apresentamos uma primeira proposta de algoritmo de enxame para optimização dos diversos aspectos da aplicação das MVS, a qual permite, dentro de alguns constrangimentos, a um utilizador fornecer apenas o conjunto de dados de treino, deixando ao cuidado do algoritmo a identificação de um núcleo adequado, possivelmente não PSD, a determinação dos parâmetros de núcleo e do parâmetro C, bem como o posterior treino evolucionário da MVS resultante.

- O algoritmo de treino e o algoritmo de optimização geral foram ambos implementados como operadores adicionais num software popular de análise de dados, permitindo uma fácil disseminação e utilização, indo assim de encontro à nossa motivação principal.

Os diversos contributos atrás listados resultaram em diversas publicações em actas de conferências internacionais convenientemente indexadas e publicadas em séries de impacto significativo, com sejam as *Lecture Notes in Computer Science* da *Spinger*. A primeira proposta de utilização de um mecanismo predador-presa para controlar o equilíbrio entre exploração local e global nos optimizadores por enxames de partículas foi publicada em [Silva et al, 2002b]. Uma comparação mais extensa com os optimizadores de enxame padrão foi apresentada em [Silva et al, 2002a] e uma versão adaptativa, capaz de evoluir os parâmetros do algoritmo simultaneamente com a optimização da função, discutida em [Silva et al, 2003].

A versão mais recente do mecanismo predador-presa, descrita nesta dissertação, foi publicada em [Silva et al, 2012a], conjuntamente com a proposta de utilização de partículas batedoras e extensos resultados experimentais. O efeito das diferentes partículas batedoras foi discutido em [Silva and Gonçalves, 2013c]. O algoritmo de enxame para treino de máquinas de vectores de suporte foi introduzido em [Silva and Gonçalves, 2013a], tendo os resultados experimentais para os diferentes núcleos não PSD sido apresentados em [Silva and Gonçalves, 2013b].

Finalmente, os conhecimentos adquiridos ao longo da realização deste trabalho, especialmente na área da inteligência de enxame, foram essenciais para o sucesso de vários trabalhos de colaboração, realizados durante o período em que aquele foi efectuado, e que também deram origem a diversas publicações [Brabazon et al, 2012; Silva et al,

2012b, 2013]. Embora não resultem directamente do trabalho descrito desta tese, estas colaborações foram, ainda assim, por ele fortemente influenciadas.

1.4 Organização da Dissertação

Após esta introdução, o restante desta dissertação encontra-se organizado da forma a seguir descrita. No próximo capítulo apresentamos de forma sumária a formulação clássica das máquinas de vectores de suporte e discutimos as consequências da utilização de funções de núcleo não positivas semi-definidas. O capítulo 3 consiste numa introdução à computação evolucionária, com particular ênfase na inteligência de enxame. No capítulo 4 resumimos as principais abordagens evolucionárias aos diversos aspectos inerentes à utilização de MVS presentes na bibliografia da área e discutimos as oportunidades de investigação em aberto. O capítulo 5 é dedicado à introdução do algoritmo de optimização predador-presa com batedores, bem como à descrição do ambiente experimental onde será testado, incluindo os diversos algoritmos utilizados para comparação e as diversas funções de teste. Os resultados experimentais obtidos pelos algoritmos de optimização e a sua discussão são apresentados no capítulo 6. No capítulo 7 descrevemos a adaptação do algoritmo genérico desenvolvido anteriormente ao problema específico do treino de máquinas de vectores de suporte e a sua integração no algoritmo global de optimização de MVS, o qual é igualmente apresentado. Também neste capítulo descrevemos o ambiente experimental onde testaremos os algoritmos, incluindo os diversos conjuntos de dados correspondentes a problemas de classificação e os algoritmos utilizados para comparação. No capítulo 8 apresentamos e discutimos os resultados obtidos pelos algoritmos de treino utilizando diversas funções de núcleo, bem como os resultados do algoritmo de optimização de MVS. Finalmente, o capítulo 9 inclui as conclusões finais retiradas deste trabalho, sendo ainda discutidas as perspectivas de trabalho futuro que o mesmo abre.

Capítulo 2

Máquinas de Vectores de Suporte

Na sua formulação mais habitual (ver, por exemplo, [Burges, 1998; Cristianini and Scholkopf, 2002; Shawe-Taylor and Cristianini, 2004]), as máquinas de vector de suporte, são mecanismos de classificação que, dado um conjunto de treino

$$T = \{(\mathbf{x}_1, y_1), ..., (\mathbf{x}_n, y_n)\}, \tag{2.1}$$

com $\mathbf{x}_i \in \mathbb{R}^m$ e $y_i \in \{\pm 1\}$, assumindo n exemplos com m atributos reais, procuram aprender um hiperplano

$$\langle \mathbf{w}, \mathbf{x} \rangle + b = 0, \tag{2.2}$$

com $\mathbf{w} \in \mathbb{R}^m$ e $b \in \mathbb{R}$, que correctamente separe todos os exemplos da classe -1 dos da classe $+1$. Utilizando este hiperplano, uma nova instância $\mathbf{x}$ é classificada de acordo com

$$f(\mathbf{x}) = \text{sgn}(\langle \mathbf{w}, \mathbf{x} \rangle + b). \tag{2.3}$$

Existem muitos métodos que permitem aprender o hiperplano em (2.2), e.g., através da regra de treino do perceptrão, mas, na sua generalidade, todos se baseiam na minimização do risco empírico, aproximado pelo erro de classificação no conjunto de exemplos de treino. Como a figura 2.1 procura ilustrar, estes métodos podem encontrar uma infinidade de hiperplanos, todos com o mesmo desempenho nos dados de treino, mas cujo desempenho poderá ser diferente quanto testados em novas instâncias.

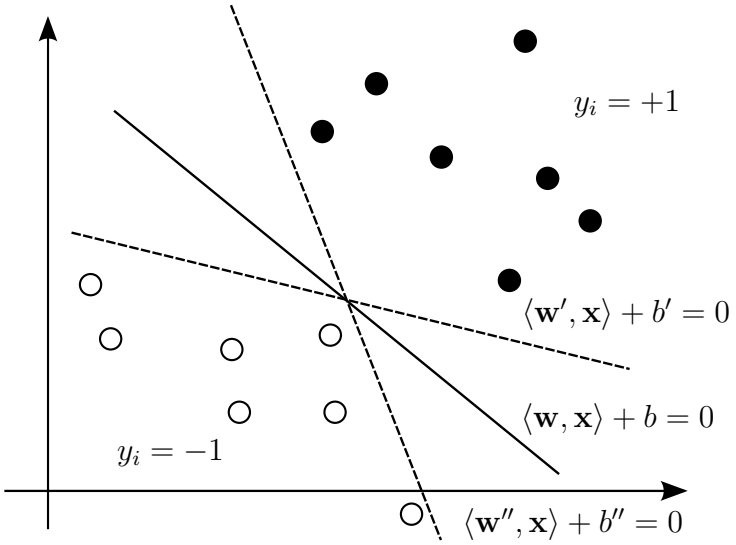


Figura 2.1: Quando os dados de treino são linearmente separáveis, existe uma infinidade de hiperplanos $\langle \mathbf{w}, \mathbf{x} \rangle + b = 0$, com diferentes valores para $\mathbf{w}$ e b, que correctamente separam todos os exemplos com classe $y_i = -1$ (círculos preenchidos a branco) dos exemplos com classe $y_i = +1$ (círculos preenchidos a negro).

2.1 MVS para Dados Linearmente Separáveis

Um dos aspectos em que as máquinas de vector de suporte se diferenciam de outros métodos de classificação é no facto de, no seu treino, não ser apenas minimizado o risco empírico. É intuitivo considerar que, de todos os hiperplanos que correctamente classificam os exemplos de treino, aquele que mais distância apresentar em relação aos exemplos mais próximos é também aquele que melhor generalizará quando novos dados lhe forem apresentados para classificação. Podemos pensar que este é o hiperplano que apresenta uma maior margem de erro para novas instâncias, já que, ao maximizar a distância às regiões ocupadas por cada classe, garante uma maior tolerância no caso da classificação de novas instâncias que se encontrem entre essa região e o hiperplano classificador.

A maximização dessa distância, ou margem, entre o hiperplano discriminador e os exemplos mais próximos, é uma característica dos chamados métodos de margem larga[1], dos quais as máquinas de vectores de suporte são uma instância. Maximizar a margem é uma forma de diminuir o chamado risco estrutural, o qual está relacionado com a qualidade da função de decisão. As máquinas de vectores de suporte procuram portanto minimizar o risco estrutural, o qual engloba não só o risco empírico, mas também uma medida da qualidade do classificador. De uma forma geral, esta abordagem procura evitar o sobre-ajustamento do classificador, dando preferência a funções de classifi-

[1]Do inglês *large margin methods*.

cação que aparentem ser mais promissoras do ponto de vista da sua capacidade de generalização para novos dados.

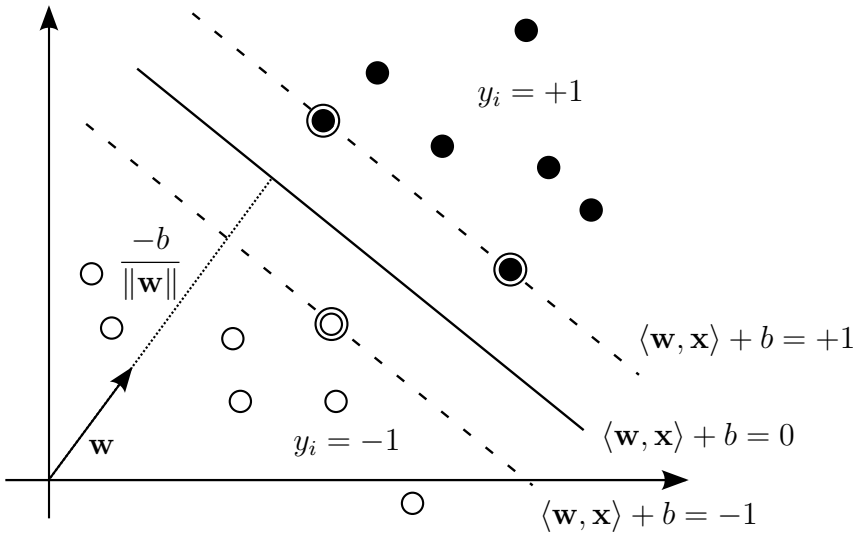


Figura 2.2: Hiperplano óptimo seleccionado para maximizar a margem, ou distância até aos exemplos mais próximos, num problema de classificação. O hiperplano é definido pelo vector $\mathbf{w}$, que lhe é perpendicular, e pelo limiar b. A distância do hiperplano à origem é dada por $\frac{|b|}{\|\mathbf{w}\|}$. Os pontos mais próximos do hiperplano (assinalados com um duplo círculo) são denominados vectores de suporte.

A figura 2.2 ilustra o hiperplano óptimo pretendido para uma máquina de vectores de suporte num problema de classificação binário. O treino ou optimização de uma máquina de vectores de suporte consiste na procura desse hiperplano de entre todos aqueles que classificam correctamente os exemplos. Para qualquer hiperplano que classifique correctamente todos os pontos do conjunto de treino $(\mathbf{x_i}, y_i)$, verifica-se que:

$$\forall i : y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) \geqslant 0. \tag{2.4}$$

Normalizando $\mathbf{w}$ e b de maneira a que os pontos mais próximos do hiperplano separador satisfaçam $|\langle \mathbf{w}, \mathbf{x} \rangle + b| = 1$, podemos reescrever a equação (2.4) como:

$$\forall i : y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) \geqslant 1, \tag{2.5}$$

Os exemplos mais próximos do hiperplano separador encontram-se, após esta normalização, sobre os hiperplanos $\langle \mathbf{w}, \mathbf{x}_i \rangle + b = 1$ e $\langle \mathbf{w}, \mathbf{x}_i \rangle + b = -1$, como se pode ver na figura 2.2. A distância respectiva destes hiperplanos à origem é dada por $\frac{|1-b|}{\|\mathbf{w}\|}$ e $\frac{|-1-b|}{\|\mathbf{w}\|}$. Como os hiperplanos são paralelos, a distância entre eles, denominada margem, é dada por $\frac{2}{\|\mathbf{w}\|}$.

Assim, o hiperplano classificador óptimo pode ser encontrado maximizando a margem $\frac{2}{\|\mathbf{w}\|}$, ou, de maneira a simplificar os cálculos posteriores, minimizando $\frac{1}{2}\|\mathbf{w}\|$ sujeito às restrições (2.5). O problema de optimização para uma MVS linear e um problema linearmente separável é então:

$$\text{minimizar } \frac{1}{2}\|\mathbf{w}\|^2, \tag{2.6}$$

$$\text{sujeito a } \forall i : y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) \geqslant 1. \tag{2.7}$$

2.2 Dados Não Separáveis

A resolução deste problema (normalmente sob uma forma dual do mesmo), permite encontrar o hiperplano óptimo em termos de margem, desde que todos os exemplos de treino sejam separáveis por esse hiperplano. No entanto, como a figura 2.3 ilustra, esse não é o caso geral, sendo necessário alterar a formulação do problema básico de maneira a lidar com dados não-separáveis.

Para tal são introduzidas variáveis de folga[2] positivas ξ_i em cada uma das restrições (2.5), de maneira a relaxar essas restrições, mas apenas quando tal é necessário, i.e., deve haver uma penalização associada a estas variáveis de folga na função a optimizar. Isto é conseguido alterando as restrições (2.5) para:

$$\forall i : y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) \geqslant 1 + \xi_i. \tag{2.8}$$

Como é possível observar na figura 2.3, para uma instância $\mathbf{x}_i$ ser classificada de forma errada, o valor da variável de folga correspondente terá de exceder 1, o que torna $\sum_i \xi_i$ num limite superior para o número de erros no conjunto de treino. Este novo termo é portanto adicionado à função a optimizar, pesado por uma constante C, dando origem ao seguinte problema de optimização:

$$\text{minimizar } \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_i \xi_i, \tag{2.9}$$

$$\text{sujeito a } \forall i : y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) \geqslant 1 - \xi_i, \tag{2.10}$$

O parâmetro C, escolhido pelo utilizador, permite controlar o equilíbrio entre a importância dada à optimização da margem e o número de erros, com um valor elevado de C correspondendo a uma maior importância do número de erros.

[2]Do inglês *slack variables*.

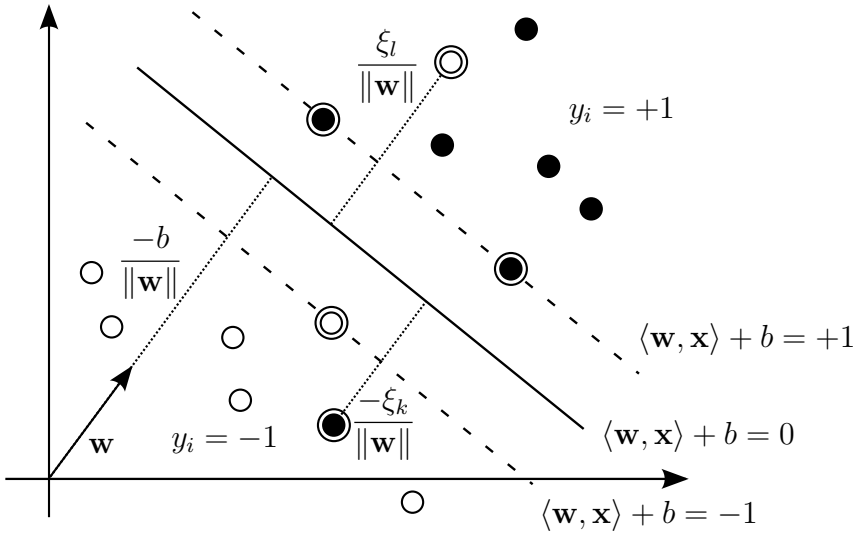


Figura 2.3: Problema de classificação não separável, com os exemplos de índice k e l do lado errado do hiperplano separador. De maneira a controlar o número de erros são associadas variáveis de folga ξ_i a cada restrição e a sua soma $\sum_i \xi_i$ adicionada à função a optimizar. Como um exemplo só é erradamente classificado quando a folga é superior a 1 (já que se encontra do lado errado do hiperplano), aquele termo é um limite superior para o número de exemplos de treino mal classificados.

Geralmente não é o problema (2.9) que é optimizado, mas sim uma versão dual, o problema dual de Lagrange, o qual implica a introdução de multiplicadores de Lagrange positivos α_i, um para cada uma das restrições (2.8). Existem duas razões importantes para assim ser. Por um lado, a transformação do problema no seu dual permite que as restrições passem a ser sobre os parâmetros de Lagrange, o que as torna mais fáceis de tratar. Por outro lado, nesta nova formulação do problema, os exemplos de treino aparecem apenas em termos de produtos vectoriais. Esta propriedade é essencial à posterior extensão das máquinas de vectores de suporte para o caso não linear e para espaços de entrada diferentes de $\mathbb{R}^m$. A nova formulação do problema de optimização é então

$$\text{maximizar } \sum_i \alpha_i - \frac{1}{2} \sum_i \sum_j \alpha_i \alpha_j y_i y_j \langle \mathbf{x}_i, \mathbf{x}_j \rangle, \tag{2.11}$$

$$\text{sujeito a } \sum_i \alpha_i y_i = 0 \tag{2.12}$$

$$\text{e } \forall i : 0 \leqslant \alpha_i \leqslant C, \tag{2.13}$$

onde as variáveis α_i são os multiplicadores de Lagrange e os pares $(\mathbf{x}_i, y_i)$, correspondem aos exemplos de treino. A optimização do problema resulta num vector solução α^* que

é utilizado na classificação de novos exemplos usando

$$f(\mathbf{z}) = \text{sgn}\left(\sum_i \alpha_i y_i \langle \mathbf{x}_i, \mathbf{z}\rangle + b\right). \tag{2.14}$$

Normalmente, no vector solução, os α_i são uma pequena percentagem do número total de instâncias, i.e., apenas um pequeno número de exemplos de treino, com $\alpha_i \neq 0$, contribui para a classificação das novas instâncias. Estes exemplos são os vectores de suporte a que a denominação deste método alude. Note-se que, mesmo na função de classificação (2.14), as instâncias são referenciadas sempre apenas através do seu produto interno. Como já foi referido, este aspecto é essencial para a extensão da metodologia anterior a problemas não lineares.

2.3 Problemas Não Lineares

A formulação das máquinas de vectores de suporte apresentada até agora funciona bem em problemas em que os exemplos são linearmente separáveis, ou, não o sendo, a sobreposição entre classes não é elevada. Não é, no entanto, viável em problemas em que a fronteira entre as classes é completamente não-linear, já que um hiperplano não poderá ser utilizado de forma eficiente, do ponto de vista do desempenho de classificação, como superfície discriminadora.

No entanto, toda a formulação em (2.11) e (2.14) assenta na utilização do produto interno entre instâncias, o qual pode ser visto como uma medida de semelhança entre as mesmas. Para esta formulação continuar a ser utilizada, e o hiperplano poder continuar como superfície de classificação, uma solução elegante consiste em considerar uma função de mapeamento

$$\Phi : \mathbb{R}^m \mapsto \mathcal{H}, \tag{2.15}$$

a qual faz corresponder cada instância do espaço de dados original $\mathbb{R}^m$ a um ponto num novo espaço com produto interno $\mathcal{H}$, onde os exemplos transformados possam ser linearmente separados. Um primeiro aspecto da elegância desta abordagem manifesta-se no facto de toda a metodologia descrita se poder manter, passando apenas a optimização a depender do produto interno das instâncias mapeadas para o novo espaço $\mathcal{H}$, i.e., utilizando $\langle \Phi(\mathbf{x}_i), \Phi(\mathbf{x}_j)\rangle$ em vez de $\langle \mathbf{x}_i, \mathbf{x}_j\rangle$.

A utilização directa da função de mapeamento apresenta, no entanto, dificuldades, que começam com a necessidade de a definir explicitamente. Essas dificuldades são

ultrapassadas com a utilização de funções de núcleo, ou núcleos, da forma

$$k(\mathbf{x}_i, \mathbf{x}_j) = \langle \Phi(\mathbf{x}_i), \Phi(\mathbf{x}_j) \rangle. \tag{2.16}$$

Qualquer função de mapeamento Φ possui um núcleo associado, o qual pode ser sempre utilizado directamente na formulação do problema de optimização em substituição do produto interno. Funções de núcleo frequentemente utilizadas incluem, por exemplo, as funções de núcleo polinomiais de grau d

$$k(\mathbf{x}_i, \mathbf{x}_j) = (\kappa \langle \mathbf{x}_i, \mathbf{x}_j \rangle + \delta)^d, \tag{2.17}$$

e a função de base radial

$$k(\mathbf{x}_i, \mathbf{x}_j) = e^{\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{\sigma}}. \tag{2.18}$$

A utilização da função de núcleo no problema de optimização permite assim às MVS lidarem com problemas não-lineares, tornando a abordagem geral para dados não separáveis e não lineares. Refira-se, mais uma vez, a elegância da generalização, a qual permite separar a remoção da não-linearidade e treino do classificador em duas fases distintas e modulares, que inclusivamente permitem a utilização de diferentes métodos de classificação, desde que baseados sempre no produto interno dos exemplos de treino. O problema de optimização final fica então

$$\text{maximizar } \sum_i \alpha_i - \frac{1}{2} \sum_i \sum_j \alpha_i \alpha_j y_i y_j k(\mathbf{x}_i, \mathbf{x}_j), \tag{2.19}$$

$$\text{sujeito a } \sum_i \alpha_i y_i = 0 \tag{2.20}$$

$$\text{e } \forall i : 0 \leqslant \alpha_i \leqslant C, \tag{2.21}$$

sendo a classificação de uma nova instância $\mathbf{z}$ feita recorrendo a:

$$f(\mathbf{z}) = \text{sgn}\left(\sum_i \alpha_i y_i k(\mathbf{x}_i, \mathbf{z}) + b \right) \tag{2.22}$$

2.4 Treino de Máquinas de Vectores de Suporte

Num problema com n exemplos, a aplicação da função de núcleo k a cada par possível de dados dá origem a uma matriz $\mathbf{K}$ com dimensões $n \times n$ em que

$$K_{ij} = k(\mathbf{x}_i, \mathbf{x}_j). \tag{2.23}$$

Se esta matriz for positiva definida, i.e., se obedecer à condição

$$\sum_{i,j} c_i c_j K_{ij} > 0, \forall c_i, c_j \in \mathbb{R}, \tag{2.24}$$

então a função objectivo é côncava (convexa no primal) e possuidora de uma única solução global. Mesmo no caso de $\mathbf{K}$ ser positiva semi-definida, obedecendo a

$$\sum_{i,j} c_i c_j K_{ij} \geqslant 0, \forall c_i, c_j \in \mathbb{R}, \tag{2.25}$$

embora possa haver mais do que uma solução, todas elas serão óptimos globais. O problema de optimização dual constitui assim um problema de programação quadrática (PQ) com uma função de optimização côncava, havendo pacotes de optimização numérica que podem ser utilizados para encontrar o óptimo.

Toda a informação necessária ao treino se encontra na matriz K, exceptuando as classificações y_i das instâncias, o que permite o cálculo *a priori* da matriz de núcleo e evita a necessidade de acesso directo aos exemplos durante a optimização. As abordagens iniciais à optimização de MVS baseavam-se na utilização de pacotes de optimização numérica capazes de resolver problemas de programação quadrática.

A resolução do problema de programação quadrática representado pela matriz K pode revelar-se bastante difícil, já que as suas dimensões são $n \times n$, sendo n o número de exemplos. Para dezenas ou centenas de milhares de exemplos torna-se impraticável manter toda a matriz em memória, tendo por isso surgido várias linhas de abordagem ao treino de MVS capazes de lidar com problemas de dimensão arbitrária. Destas linhas podemos destacar três mais relevantes, as quais deram origem a algumas das implementações mais populares entre os utilizadores de MVS.

2.4.1 Abordagens baseadas em *chunking*

Os métodos denominados *chunking*, inicialmente propostos por Vapnik, assentam no facto da remoção das linhas e colunas de **K** correspondentes aos multiplicadores de Lagrange com valor zero não alterarem a solução do problema. O problema global de programação quadrática pode assim ser dividido numa série de subproblemas mais pequenos, até que todos os α_i com valor não zero sejam identificados. A cada passo, o subproblema a resolver engloba todos os exemplos com multiplicadores não zero do passo anterior e os exemplos que violam mais significativamente as condições de optimalidade.

A complexidade computacional do problema é assim reduzida do quadrado do número de exemplos para aproximadamente o quadrado do número de exemplos com multiplicadores não zero. O sucesso desta abordagem assenta no facto do número de vectores de suporte (logo, multiplicadores não zero) ser relativamente pequeno em relação ao número total de exemplos. Mesmo assim, para problemas muito grandes, pode não ser possível resolver os problemas de PQ resultantes, devido à incapacidade de manter as matrizes em memória.

2.4.2 Abordagens baseadas em decomposição

As estratégias baseadas em decomposição também apostam na divisão do problema total numa sequência de problemas mais pequenos, embora consigam trabalhar com problemas intermédios de menor dimensão do que as abordagens baseadas em *chunking*. Na abordagem proposta em [Osuna et al, 1997a,b] é demonstrado que, desde que pelo menos um exemplo que viole as condições de optimalidade seja adicionado aos exemplos do subproblema anterior, cada nova iteração melhora o valor da função a optimizar, simultaneamente garantindo a manutenção da viabilidade da solução. O algoritmo resultante irá assim convergir, desde que a sequência de subproblemas vá sempre incluindo novas instâncias que violem as condições de optimalidade. A proposta original sugere a utilização de subproblemas de tamanho fixo, permitindo assim a resolução de problemas de dimensão arbitrária.

Joachims [1999] apresenta outra abordagem baseada em decomposição. Neste método a escolha dos exemplos para o subproblema é feita utilizando o gradiente da função a optimizar. A cada iteração, o subproblema é assim definido utilizando a direcção de maior declive da função, acelerando a optimização do problema principal. Esta abordagem deu origem a uma implementação muito popular, denominada SVM^{Light}, da qual existem várias variantes, sendo mySVM uma das mais utilizadas [Rüping, 2000].

2.4.3 Optimização sequencial mínima

Ambas as abordagens anteriores necessitam de um pacote de software capaz de optimizar numericamente um problema de programação quadrática, sendo que este processo, além de demorado, apresenta as suas próprias dificuldades. A optimização sequencial mínima (OSM), proposta por Platt [1999], ultrapassa esta limitação ao proceder a uma decomposição do problema principal, nos mesmos moldes das abordagens baseadas em decomposição, mas resolvendo a cada iteração o subproblema mínimo, i.e., subproblemas com apenas duas instâncias.

A cada iteração, este método selecciona dois exemplos (multiplicadores) para optimizar, executa essa optimização de forma analítica e actualiza a MVS utilizando os novos valores encontrados. A necessidade de um pacote adicional de optimização numérica para a resolução dos sub-problemas de PQ é a assim ultrapassada. Embora esta abordagem implique a resolução de um muito maior número de sub-problemas, como esta é feita de forma analítica e é muito rápida, pode mesmo assim resultar num elevado ganho computacional em relação aos outros métodos, especialmente por não haver necessidade de armazenar grandes matrizes em memória.

Um implementação baseada em OSM frequentemente utilizada em trabalhos experimentais é apresentada por Chang and Lin [2001], sendo conhecida como LIBSVM. Esta implementação usa o método de decomposição proposto por Fan et al [2005], o qual apresenta melhorias em relação às abordagens anteriores em termos da selecção do conjunto de trabalho, acelerando assim a convergência do algoritmo.

2.5 Treino de MVS com Núcleos Indefinidos

As máquinas de vectores de suporte, utilizando os algoritmos de treino mencionados na secção anterior, têm sido aplicadas com sucesso às mais diversas áreas. No entanto, esse sucesso está dependente da utilização de uma função de núcleo positiva semi-definida. Quando a função de núcleo é indefinida, o problema de optimização resultante é multimodal [Haasdonk, 2005]. Embora as implementações mencionadas possam ser aplicadas a problemas em que a matriz K não é positiva semi-definida, não estão, no entanto, preparadas para lidar com a possibilidade de existência de múltiplos óptimos locais. Poderão assim ficar presas nesses óptimos, produzindo soluções de qualidade inferior, ou mesmo, em alguns casos, não terminar a execução.

A possibilidade de treinar MVS com núcleos indefinidos é, no entanto, desejável por vários motivos:

- A prova em como uma função de núcleo é positiva definida pode ser complexa,

para alguns núcleos, especialmente na sua aplicação prática por utilizadores não familiarizados com a área.

- Há funções de núcleo interessantes, como o núcleo da tangente hiperbólica, ou núcleo sigmóide, usado frequentemente em redes neuronais, que são comprovadamente indefinidas [Lin and Lin, 2003].
- Do ponto de vista experimental, alguns destes núcleos apresentam resultados bastante competitivos [Bahlmann et al, 2002; Decoste and Schölkopf, 2002; Haasdonk and Keysers, 2002; Moreno et al, 2003].
- A investigação na área da aprendizagem da função de núcleo teria vantagens na eliminação da restrição a núcleos positivos definidos, já que existem métodos (inclusivamente alguns evolucionários) que não garantem que os núcleos aprendidos sejam PSD [Howley and Madden, 2005].
- Determinados métodos de síntese de novos núcleos, como a substituição de medida de distância num núcleo convencional, podem resultar em núcleos não PSD [Haasdonk and Bahlmann, 2004].
- Existem áreas com métodos bem definidos de geração de medidas de similaridade que não resultam em núcleos positivos semi-definidos, nomeadamente, na área da biologia computacional, as medidas de similaridade para sequências proteicas baseadas nos algoritmos de Smith-Waterman e BLAST.

Esta é uma área activa e actual de pesquisa, existindo várias abordagens recentes que permitem lidar com núcleos não positivos semi-definidos [Chen et al, 2009]. Uma linha possível de abordagem consiste em alterar $\mathbf{K}$ de maneira a que passe a ser positiva semi-definida, existindo para tal diversas metodologias (ver, por exemplo, [Wu et al, 2005]). A desvantagem imediata destas abordagens consiste no facto de alguma da informação contida no núcleo original se perder durante a transformação.

Um método relacionado consiste na aprendizagem da matriz de núcleo simultaneamente com o treino da MVS, considerando a matriz original como uma aproximação ruidosa de um verdadeiro núcleo positivo semi-definido [Chen and Ye, 2008; Luss and d'Aspremont, 2009]. Esta abordagem é mais complexa computacionalmente por implicar a aprendizagem simultânea da matriz de núcleo e a sua consequente manutenção em memória. Além disso, também aqui alguma informação do núcleo original se perde (o presumível "ruído"). Estas duas linhas de abordagem têm em comum o facto de manterem a formulação das máquinas de vectores de suporte, sendo os núcleos adaptados de maneira a que essa formulação seja válida.

Outras abordagens procuram usar directamente os núcleos indefinidos e toda a informação neles contida. A dificuldade central a essa utilização consiste no facto de um

núcleo indefinido não corresponder a um produto interno no espaço induzido. A formulação básica das máquinas de vectores de suporte é assim difícil de interpretar e a função a optimizar passa a ser multimodal, o que apresenta óbvias dificuldades para os algoritmos de treino utilizados [Haasdonk, 2005].

Uma forma de ultrapassar estas dificuldades consiste em desenvolver uma nova formulação e métodos de treino para os espaços induzidos por núcleos indefinidos [Ong et al, 2004]. A aprendizagem nestes novos espaços é no entanto muito mais complexa, o que torna inviável a sua utilização em problemas de dimensão útil.

Haasdonk [2005] nota que muitos autores têm optado pela abordagem "heurística" de utilizar núcleos indefinidos directamente na formulação tradicional das máquinas de vectores de suporte. Os resultados empíricos dessa utilização são frequentemente muito bons, embora as bases teóricas percam solidez. O mesmo autor apresenta uma nova interpretação da aprendizagem com núcleos indefinidos como sendo a minimização da distância entre *hulls* convexos num espaço pseudo-euclidiano.

O aspecto mais interessante desta interpretação consiste no facto de as máquinas de vectores de suporte poderem corresponder a soluções com uma interpretação geométrica bem definida dentro desta nova formulação, sendo a maior dificuldade o facto dos algoritmos de treino comuns não estarem normalmente preparados para lidar com os problemas multimodais resultantes dos núcleos indefinidos. Tanto Haasdonk [2005] como, anteriormente, Lin and Lin [2003], concluem que alguns métodos clássicos podem ser utilizados para treinar SVM com núcleos não definidos, podendo, no entanto, convergir para soluções sub-óptimas da função objectivo, com repercussões óbvias em termos do desempenho dos classificadores correspondentes.

No nosso trabalho levamos em conta a interpretação proposta por Haasdonk [2005], bem como a constatação da qualidade dos resultados empíricos já obtidos com núcleos não PSD, para propor algoritmos evolucionários para o treino de MVS capazes de lidar com o problema de optimização global resultante da integração desses núcleos em MVS.

Capítulo 3

Computação Evolucionária

A computação evolucionária[1] é um ramo da inteligência artificial ou, mais especificamente, da inteligência computacional [Engelbrecht, 2007], que se centra no estudo e desenvolvimento de algoritmos baseados na exploração de princípios da evolução por selecção natural, inicialmente introduzidos por Darwin [1872]. Os algoritmos pertencentes a esta área têm, nas últimas décadas, sido aplicados com sucesso a uma enorme variedade de problemas complexos, incluindo problemas de optimização global [Weise, 2009], difíceis de abordar com outras técnicas.

Central à teoria de Darwin [1872] é a consideração do indivíduo como elemento de uma população que existe num determinado ambiente. É no contexto da população que a qualidade de um indivíduo pode ser avaliada, por comparação com os restantes elementos do grupo e em termos da sua adaptação ao ambiente em que vive. É também nesse contexto que os quatro princípios essenciais propostos por Darwin como centrais à evolução por selecção natural promovem a evolução de indivíduos progressivamente melhor adaptados ao seu ambiente. Esses princípios são a **variação**, a **sobrevivência e reprodução diferenciais**, a **herança** e, finalmente, o **crescimento populacional**.

O primeiro princípio da teoria de Darwin, a variação, é facilmente observável em qualquer população na natureza. Por muito homogéneo que um grupo seja, existem sempre indivíduos mais rápidos ou mais lentos, de cor mais clara ou mais escura, maiores ou menores... Indivíduos com características mais favoráveis, i.e., mais bem adaptados ao seu ambiente, poderão viver mais tempo, afugentar indivíduos competidores com maior facilidade e mesmo atrair mais e melhores parceiros sexuais. Características mais vantajosas permitirão assim a um indivíduo produzir, estatisticamente, mais descendentes, levando à sobrevivência e reprodução diferenciais dos indivíduos dentro de uma população.

[1] Do inglês *evolutionary computation*.

O facto de indivíduos mais aptos serem capazes de produzir mais descendentes seria inútil do ponto de vista da evolução por selecção natural se não fosse possível a um indivíduo passar as suas características aos seus descendentes, i.e., se não fosse possível a herança, por parte de uma nova geração, das qualidades que permitiram aos seus progenitores possuírem uma vantagem competitiva na população.

Hoje sabemos que esta herança é possível devido à codificação das características de um indivíduo no seu material genético, o qual é passado aos descendentes durante a reprodução. Nesta são aplicados dois mecanismos essenciais, a **recombinação** e a **mutação**, frequentemente imitados na computação evolucionária. Para gerar o material genético de um descendente, o material genético (os cromossomas) dos progenitores é recombinado, de forma que o novo indivíduo herda um subconjunto das suas características de cada um dos progenitores. Durante este processo podem ocorrer pequenos erros, denominados mutações, os quais permitem ligeiras variações no material genético dos filhos em relação ao material genético dos pais. É o facto da reprodução incluir o uso destes (e outros) mecanismos que garante a variação na nova geração, em vez de uma uniformidade que impossibilitaria a evolução.

O princípio final, o crescimento populacional, tem simplesmente a ver com o facto de, usualmente, qualquer população produzir mais descendentes do que os necessários para manter o seu tamanho estável. Assim, o número de indivíduos de uma população tende a crescer ao longo do tempo, desde que os recursos do ambiente o permitam. No entanto, num ambiente natural com recursos limitados (território, alimentos...), o tamanho da população não pode aumentar indefinidamente.

A escassez de recursos leva à selecção natural central à teoria da evolução, já que a competição por esses recursos deverá, também estatisticamente, favorecer os indivíduos mais aptos, i.e., com melhores características, os quais viverão mais tempo e produzirão mais descendentes, que por sua vez herdarão essas mesmas características vantajosas. A aplicação destes mecanismos ao longo do tempo resulta assim num aumento do nível médio de adaptação da população ao seu ambiente. Note-se que todo este processo age ultimamente sobre os cromossomas que codificam a informação genética de um indivíduo (o seu **genótipo**), em vez de directamente sobre as suas características físicas (o seu **fenótipo**).

A computação evolucionária baseia-se numa metáfora computacional que segue de perto os elementos descritos para a evolução natural. O ambiente é substituído por um problema a resolver e os indivíduos correspondem a soluções candidatas. O nível de adaptação é medido por uma função de avaliação ou desempenho, a qual é definida de maneira a reflectir a qualidade da solução candidata (indivíduo) recebida como argumento. Operadores de recombinação e mutação são implementados de maneira a reflectir os mecanismos correspondentes na evolução natural. Finalmente, a utilização de uma população (conjunto de soluções candidatas) de tamanho fixo, associada a um

operador de selecção que garante uma maior probabilidade de reprodução (e consequente maior número de descendentes) a indivíduos com melhor desempenho, reflecte a pressão da selecção natural e a sobrevivência dos mais aptos.

O facto dos algoritmos da área da computação evolucionária permitirem optimizar uma função de desempenho que mede a qualidade das soluções para um dado problema, procurando assim uma solução óptima para o mesmo, coloca-os na área mais vasta dos algoritmos de optimização global, a qual descreveremos brevemente na secção seguinte. Posteriormente, dedicaremos uma secção à apresentação de um algoritmo evolucionário (AE) genérico, descrevendo por fim, nas secções restantes, as variantes de algoritmo evolucionário mais relevantes, tanto para a área a que pertencem como para o nosso trabalho.

3.1 Optimização Global

A optimização global é a área científica que se preocupa com a procura de soluções óptimas para determinado problema. A necessidade de uma solução óptima coloca-se tanto na natureza, como na generalidade das actividades humanas. Problemas típicos da área incluem a previsão da estrutura de proteínas (pela minimização de uma função de energia), a gestão de portfólios, o design optimizado de equipamentos e estruturas, o planeamento de terapia do cancro e inúmeras outras aplicações em que a qualidade da solução pode ser medida por um ou mais critérios.

Para encontrar a solução óptima (ou soluções) para determinado problema, o primeiro passo consiste em definir o espaço $\mathbb{X}$ de todas as soluções possíveis do problema. Para haver optimização é necessário poder comparar a qualidade destas soluções entre si de acordo com critérios relevantes para o problema em causa. Estes critérios são codificados num conjunto de funções $\mathbf{f} = f_1, f_2, ..., f_n$, normalmente denominadas funções objectivo. Uma função objectivo $f_i : \mathbb{X} \mapsto \mathbb{Y}$ mapeia as soluções do problema em valores reais $y = f_i(x) \in \mathbb{Y} \subset \mathbb{R}$, sendo que o valor de y mede a qualidade da solução x segundo o critério implementado por f_i. Nesta dissertação interessa-nos sobretudo o caso em que $|\mathbf{f}| = 1$, i.e., o caso particular em que a qualidade de uma solução é medida por apenas um critério. Quando $|\mathbf{f}| > 1$ estamos na presença de um problema de optimização multi-critério, mais geral mas menos relevante para este trabalho.

Um algoritmo de optimização global tem assim como objectivo encontrar os elementos $x^* \in \mathbb{X}$ que, de acordo com os valores $y = f(x)$ da função objectivo, são soluções óptimas do problema. Do ponto de vista matemátio, o algoritmo procura os óptimos globais de $f(x)$, os quais podem ser máximos globais ou mínimos globais, conforme se trate de um problema de maximização ou de um problema de minimização. Um óptimo global é um máximo global $\hat{x}$ quando $f(\hat{x}) \geqslant f(x), \forall x \in \mathbb{X}$. Quando $f(\check{x}) \leqslant$

$f(x), \forall x \in \mathbb{X}$ o óptimo global é um mínimo global $\breve{x}$.

Se num dado problema existir um único óptimo, existem algoritmos eficientes, normalmente baseados no seguimento do gradiente da função objectivo, que permitem encontrá-lo com facilidade. O grande desafio da optimização global surge quando existem vários (possívelmente muitos) óptimos, ditos locais, que, sendo a melhor solução dentro de uma vizinhança ϵ não são a melhor solução global do problema. Estas soluções são chamadas máximos locais se $f(\hat{x}_l) \geqslant f(x), \forall x \in \mathbb{X}, |x - \hat{x}_l| < \epsilon$ e mínimos locais se $f(\breve{x}_l) \leqslant f(x), \forall x \in \mathbb{X}, |x - \breve{x}_l| < \epsilon$.

Obviamente que, na presença de óptimos locais, os algoritmos baseados no seguimento do gradiente da função objectivo não garantem que encontram o óptimo global, já que podem estar simplesmente a seguir o gradiente na direcção de um máximo (ou mínimo) local. A optimização global é assim um problema bastante mais complexo do que a optimização local, para o qual existem muitas abordagem e diferentes algoritmos, os quais podem, de forma geral, ser divididos entre abordagens deterministas e abordagens estocásticas.

Quando um problema é bem compreendido e o espaço de procura não é muito grande, é frequentemente possível definir heurísticas que permitem a abordagens deterministas, como a procura no espaço de estados, encontrar soluções óptimas. Muitos problemas, no entanto, possuem espaços de procura demasiado grandes ou complexos para serem eficazmente explorados de forma determinista. As abordagens estocásticas (como por exemplo a recristalização simulada[2]) têm demonstrado ser ferramentas viáveis para este tipo de problemas.

Nesta dissertação interessa-nos sobretudo um grupo de algoritmos de optimização estocásticos que se agrupam genericamente numa área denominada computação evolucionária. Estes algoritmos são normalmente classificados como meta-heurísticas, já que tipicamente utilizam uma heurística genérica, frequentemente de inspiração biológica, como a evolução por selecção natural, como guia para o processo de procura da solução óptima. O termo "meta" da denominação decorre de, para cada problema, a heurística genérica ser combinada com a função objectivo e outro conhecimento específico, daí resultando uma instância da heurística particularmente adequada ao problema. Na próxima secção descrevemos um algoritmo evolucionário genérico, bem como os elementos que têm de ser definidos para implementar uma versão concreta e adaptada a um problema específico. Para uma discussão detalhada sobre a área da optimização global e, em especial, sobre a optimização global baseada em computação evolucionária, ver [Weise, 2009].

[2]Do inglês *simulated annealing*.

3.2 O Algoritmo Evolucionário Genérico

O desenvolvimento de um algoritmo evolucionário, e a sua aplicação a um problema de optimização concreto, implicam uma imitação computacional dos princípios e mecanismos que identificámos anteriormente como centrais à evolução por selecção natural. Nas próximas subsecções discutimos os diversos elementos que é necessário definir, incluindo os operadores e funções genéricos necessários e os espaços por eles manipulados. Após essa discussão apresentaremos um algoritmo evolucionário genérico resultante da agregação desses elementos.

3.2.1 Representação

O primeiro passo para a aplicação de um algoritmo evolucionário consiste na escolha de uma representação computacional adequada para as soluções candidatas, denominadas indivíduos. Esta representação, que pode ir desde o simples vector binário até estruturas de dados de elevada complexidade, define o espaço de procura $\mathbb{G}$. Este é constituído por todas as soluções g possíveis de construir utilizando a representação definida. A importância da escolha da representação fica aqui bem vincada, já que apenas soluções que nela possam ser expressas poderão ser alvo do processo de procura.

Imitando os mecanismos biológicos que inspiram esta abordagem, a codificação g de uma solução é denominada **genótipo** e pode normalmente ser dividida em várias partes, ou genes, codificando cada um uma característica da solução. Os genes podem estar agrupados em um (o mais comum) ou vários conjuntos, denominados **cromossomas**. A posição de um gene no cromossoma é o seu **locus** e os valores que cada gene pode tomar (e.g., 0 ou 1 num gene binário) são os seus **alelos**. Também estas denominações foram adoptadas a partir de equivalentes biológicos.

A metáfora biológica estende-se também ao facto de, no caso geral, ter de haver um mecanismo que faça o mapeamento entre a codificação $g \in \mathbb{G}$ de um indivíduo e a solução propriamente dita, $x \in \mathbb{X}$, em que $\mathbb{X}$ é o chamado espaço do problema. Este espaço contém todas as soluções candidatas para o problema original. No caso biológico, g corresponde ao material genético de um indivíduo, enquanto x corresponderia ao organismo propriamente dito, também denominado **fenótipo**. Num problema de optimização computacional (e.g., optimização de uma lente fotográfica), g corresponde à estrutura de dados utilizada para codificar a solução (e.g., vector de parâmetros reais) e x à solução descodificada (e.g., a lente fotográfica construída a partir dos parâmetros). No entanto, ao contrário do que acontece na biologia, na computação evolucionária podem ocorrer problemas em que $\mathbb{G}$ e $\mathbb{X}$ coincidem, por exemplo em problemas de optimização de funções, nos quais g e x são o mesmo vector de parâmetros.

Uma característica comum a todos os algoritmos da área da computação evolucionária tem a ver com o facto de todos manterem um conjunto de indivíduos, denominado **população**. Ao contrário da maioria dos outros algoritmos de optimização global, os algoritmos evolucionários fazem assim uma exploração paralela do espaço de procura. A população é tipicamente armazenada num vector $\mathbf{p}$, no qual cada p_i corresponde a um indivíduo, normalmente representado por um tuplo $\langle g_i, v_i \rangle$, em que g_i é o genótipo do indivíduo e v_i um valor real que reflecte a sua qualidade. Há, no entanto, representações em que outros elementos são mantidos no tuplo, como por exemplo um g'_i correspondente a uma solução particularmente promissora encontrada pelo indivíduo no passado (memória) ou mesmo a descodificação do indivíduo no espaço do problema, x_i.

Nos algoritmos evolucionários existem dois operadores importantes que, além de dependerem fortemente da representação, são essenciais para a criação da população e mapeamento dos genótipos nos fenótipos:

- iniciar() - O operador **iniciar** devolve uma população de indivíduos, inicializando os seus genótipos de acordo com a representação definida. A inicialização é normalmente aleatória, amostrando $\mathbb{G}$ de forma uniforme. É também possível utilizar conhecimento sobre o problema para introduzir directamente indivíduos mais promissores na população inicial.
- mapear(g) - O operador **mapear** recebe um genótipo g e devolve o correspondente fenótipo x. Idealmente implementa uma função de domínio $\mathbb{G}$ e contradomínio $\mathbb{X}$, fazendo o mapeamento entre os dois espaços. No entanto, dependendo do problema e do algoritmo implementado, pode nem sempre ser possível criar um elemento de $\mathbb{X}$ a partir de certos genótipos.

3.2.2 Avaliação

A partir do momento em que é possível criar indivíduos, a próxima etapa num algoritmo evolucionário consiste em medir a qualidade desses mesmos indivíduos. No caso geral, como vimos na discussão sobre a optimização global, podem haver vários critérios simultâneos de avaliação de um indivíduo. Estes critérios, implementados por várias funções f_j, medem normalmente aspectos do fenótipo do indivíduo. Em situações raras também podem ser tidas em conta características do genótipo, como a sua parsimónia. As diversas funções objectivo devolvem um valor real $y_j = f_j(x)$ para cada aspecto avaliado, mapeando assim cada indivíduo num vector $\mathbf{y} \in \mathbb{Y} \subseteq \mathbb{R}^n$, onde $\mathbb{Y}$ é denominado espaço dos objectivos.

Num algoritmo evolucionário, a avaliação de um indivíduo consiste num único valor real $v \in \mathbb{V} \subseteq \mathbb{R}$, chamando-se normalmente a $\mathbb{V}$ espaço do desempenho. O valor de

desempenho v mede a qualidade de um indivíduo em relação aos restantes indivíduos da população. No caso geral multi-critério, descrito atrás, torna-se necessário um mecanismo que permita calcular o valor de v a partir do vector $\mathbf{y}$, i.e., que permita mapear $\mathbb{Y}$ em $\mathbb{V}$. Na área da optimização multi-critério utilizam-se para tal estratégias sofisticadas de ordenação, como a relação de Pareto. Na área da computação evolucionária usa-se frequentemente a estratégia mais simples de fazer $v = \sum_{j=1}^{n} w_j f_j(x)$, ou seja v é calculado como uma soma pesada dos valores das funções objectivo, com os pesos a permitirem controlar a importância dos diferentes aspectos avaliados. No caso mais simples temos apenas uma função objectivo e $v = f(x)$.

Uma avaliação adequada dos indivíduos é essencial, já que é normalmente a única informação sobre o problema utilizada para guiar o processo de procura. Quão mais precisamente a função de desempenho for capaz de diferenciar entre a qualidade dos diferentes indivíduos, melhor será o desempenho do algoritmo. O mecanismo de avaliação é implementado recorrendo a:

- Funções objectivo $f_j(x)$ - Cada $f_j(x)$ mede um aspecto do fenótipo x relevante para a avaliação final do indivíduo. No caso mais simples apenas um aspecto é relevante e a função objectivo é $y = f(x)$.

- avaliar($\mathbf{p}$) - O operador **avaliar** recebe uma população $\mathbf{p}$ de indivíduos e armazena no campo v_i de cada tuplo p_i um valor de avaliação da solução correspondente. Tipicamente, cada indivíduo é avaliado utilizando primeiro o operador mapear para produzir o fenótipo x_i a partir do genótipo g_i, calculando de seguida os valores y_i das diversas funções objectivo e, finalmente, utilizando estes para calcular o valor final de avaliação v_i.

3.2.3 Variação

Quando discutimos os elementos essenciais da evolução por selecção natural, a variação entre indivíduos foi um dos aspectos destacados. Uma das formas dessas variação ser integrada nos algoritmos evolucionários é através da manutenção de uma população de diferentes indivíduos ou soluções. Mas para a evolução por selecção natural (e os algoritmos nela inspirados) funcionar é também necessário criar novos indivíduos diferentes dos já existentes, já que só assim se torna possível explorar o espaço de procura $\mathbb{G}$, definido pela representação adoptada.

Embora sejam utilizados muitos mecanismos de variação diferentes nos algoritmos evolucionários, na generalidade podem ser divididos em dois grandes grupos, conforme o tipo de inspiração biológica que utilizam. O primeiro grande grupo de operadores baseia-se na reprodução assexuada dos seres vivos, i.e., quando um ser vivo, como por

exemplo uma bactéria, se reproduz criando uma cópia de si próprio. Os dois aspectos essenciais deste tipo de operador são o facto de apenas o genótipo g_i de um indivíduo estar envolvido, i.e., existe um só progenitor, e a necessidade de haver obrigatoriamente erros na cópia para poder haver diversidade e evolução. Estes tipo de operadores são denominados operadores de **mutação**.

O contributo essencial dos operadores de mutação num algoritmo evolucionário tem a ver com o facto de estes serem os operadores que permitem inserir na população genes que nela não existem. Um operador de mutação permite assim manter um mecanismo de exploração global do espaço de procura, mesmo quando a generalidade da população já se encontra concentrada na vizinhança de um óptimo e a variedade genética é limitada. A probabilidade de aplicação deste tipo de operador é normalmente baixa, já que a introdução de demasiadas mutações num mesmo indivíduo leva obrigatoriamente à perda de genes benéficos por troca com genes gerados aleatoriamente.

Os operadores do segundo grande grupo inspiram-se na reprodução sexuada e são chamados operadores de **recombinação**. Estes têm como objectivo, tal como o nome indica, criar o material genético de um ou mais descendentes (normalmente dois) através da recombinação do material genético dos progenitores (também geralmente dois). Este tipo de operadores caracteriza-se por envolver sempre mais do que um progenitor e por não introduzir novo material genético na população. Assim, a sua contribuição para o algoritmo consiste na exploração local do espaço de procura na vizinhança dos indivíduos já existentes, através da (re)combinação das suas características nos novos indivíduos gerados.

A utilização deste tipo de operadores garante a existência da **herança** necessária ao funcionamento da evolução, já que em ambos os grupos é garantido que uma parte substancial dos genes de cada progenitor é passada aos descendentes. Um algoritmo evolucionário pode utilizar um ou mais operadores de cada grupo numa fase de reprodução que cria novos indivíduos a partir de um grupo de progenitores. No nosso algoritmo genérico implementamos essa fase utilizando os seguintes operadores:

- mutar(g) - Operador que recebe um único genótipo, o progenitor, e devolve um descendente que resulta da alteração aleatória de parte do progenitor. A versão mais comum percorre todo o cromossoma, podendo, com probabilidade muito baixa, substituir o alelo de cada gene por um outro alelo escolhido aleatoriamente.
- recombinar(g_1, g_2) - O operador de recombinação recebe tipicamente dois genótipos, correspondendo a dois progenitores diferentes, e devolve também tipicamente dois descendentes. Uma variação comum segue o modelo biológico de recombinação, criando os cromossomas dos descendentes através do cruzamento dos cromossomas originais em torno de um ponto escolhido aleatoriamente. Chama-se por isso recombinação de um ponto.

- variar(**p**) - Operador que no nosso algoritmo genérico é responsável pela criação de uma nova geração de soluções a partir de uma população de progenitores escolhidos para o efeito e frequentemente denominada população de reprodução. Este operador usa uma ou mais instâncias dos operadores **mutar** e **recombinar** atrás descritos para criar os genótipos nos novos indivíduos - novos pontos no espaço de procura - a partir do material genético dos progenitores.

3.2.4 Selecção

A ideia de selecção é central na computação evolucionária, agregando os princípios darwinianos do **crescimento populacional** e da **reprodução e sobrevivência diferencial**. Nos algoritmos evolucionários a estratégia de selecção é responsável por determinar quantos descendente deve produzir cada indivíduo. Este mecanismo é frequentemente utilizado para garantir que um indivíduo com melhor desempenho tem estatisticamente uma maior probabilidade de produzir descendentes do que um indivíduo com desempenho inferior. Em alguns algoritmos, por exemplo, a percentagem de descendentes esperados na geração seguinte para um determinado indivíduo é igual ao quociente da divisão do seu desempenho pelo desempenho total da população.

Após os indivíduos serem seleccionados para reprodução, uma nova geração de descendentes é criada recorrendo aos operadores de variação atrás descritos. Dá-se aqui o **crescimento populacional** a que aludia Darwin, já que a soma do número de progenitores e descendentes é superior ao tamanho da população mantida pelo algoritmo. É assim necessária uma estratégia de substituição, a qual consubstancia uma nova oportunidade de selecção, já que é possível, por exemplo, escolher apenas os melhores indivíduos de entre progenitores e descendentes para constituir a nova população. No entanto, em algoritmos em que já há selecção efectiva aquando da reprodução, o mais comum é a estratégia de substituição ser geracional, i.e., os descendentes substituem completamente os progenitores na nova população.

O papel da selecção na procura consiste em fazer a população convergir para as regiões do espaço de procura com desempenho mais elevado. Assim, é necessário algum cuidado com a pressão selectiva usada, já que uma vantagem demasiado grande para os melhores indivíduos poderá levar o algoritmo a convergir de forma demasiado rápida (convergência prematura) para um óptimo local. No algoritmo genérico a selecção é implementada por dois operadores:

- seleccionar(**p**) - Este operador recebe uma população de indivíduos e devolve uma população temporária destinada à reprodução. Esta população pode ter várias cópias de um mesmo indivíduo da população base, reflectindo uma maior probabilidade deste se reproduzir em resultado de um desempenho superior. Sobre

esta população temporária (população de reprodução) vai agir o mecanismo de variação já descrito.

- substituir($\mathbf{p}', \mathbf{p}$) - O operador **substituir** recebe a população $\mathbf{p}$ de progenitores e os descendentes $\mathbf{p}'$ criados pelos operadores de variação (recombinação e mutação). Estes são inseridos na população principal $\mathbf{p}$ utilizando um mecanismo de substituição que determina quais os progenitores que serão eliminados para dar lugar aos novos indivíduos. A nova população daí resultante é devolvida pelo operador.

3.2.5 O algoritmo

O algoritmo evolucionário genérico (algoritmo 3.1) pode agora ser descrito recorrendo aos elementos atrás apresentados. Trata-se de um algoritmo iterativo, em que a uma primeira fase de inicialização se segue um ciclo principal que processa uma população $\mathbf{p}$ de indivíduos a cada iteração até uma condição de paragem pré-definida ser atingida.

Na fase de inicialização é criada uma população inicial $\mathbf{p}$ para a iteração $t = 0$, recorrendo ao operador iniciar. Esta primeira população é então avaliada utilizando o operador avaliar($\mathbf{p}$), sendo os valores de desempenho armazenados no tuplo correspondente a cada indivíduo p_i. Após a avaliação da população original, o algoritmo entra no seu ciclo principal, o qual terminará com a satisfação de uma condição de paragem, aqui representada pelo operador terminar($t, \mathbf{p}$). Este devolve o valor lógico *verdadeiro* quando a condição implementada se verificar. As condições de paragem mais comuns são a obtenção de um valor predefinido de desempenho, a execução de um determinado número de iterações (normalmente denominadas gerações) ou a passagem de um limite de gerações sem incremento do desempenho do melhor indivíduo.

O ciclo principal do algoritmo começa com a criação de uma população temporária $\mathbf{p}'$ com cópias dos indivíduos da população principal $\mathbf{p}$ escolhidos pelo mecanismo de selecção seleccionar($\mathbf{p}$) como progenitores das próxima geração de soluções candidatas. Esta população temporária é alterada pelo mecanismo de variação variar($\mathbf{p}'$), o qual pode incluir várias instâncias de operadores de mutação e recombinação, dando assim origem a uma população $\mathbf{p}''$ de descendentes. Estes são inseridos na população principal utilizando o mecanismo de substituição substituir($\mathbf{p}'', \mathbf{p}$), criando-se assim uma nova população principal $\mathbf{p}$. O algoritmo avança então para a próxima iteração (geração $t + 1$), repetindo-se todo o processo até a condição de paragem ser satisfeita. A execução deste processo geração após geração irá, tendencialmente, melhorar o desempenho médio da população de soluções, da mesma forma que a selecção natural aumenta o nível de adaptação de uma população ao seu ambiente.

Embora até aqui tenhamos descrito os algoritmos evolucionários de forma genérica, na

Algoritmo 3.1 Algoritmo evolucionário genérico.

$t \leftarrow 0$
$\mathbf{p} \leftarrow \text{iniciar}(\mathbf{p})$
$\mathbf{p} \leftarrow \text{avaliar}(\mathbf{p})$
enquanto não terminar$(t, \mathbf{p})$ **faz**
  $\mathbf{p}' \leftarrow \text{seleccionar}(\mathbf{p})$
  $\mathbf{p}'' \leftarrow \text{variar}(\mathbf{p}')$
  $\mathbf{p} \leftarrow \text{substituir}(\mathbf{p}'', \mathbf{p})$
  $\mathbf{p} \leftarrow \text{avaliar}(\mathbf{p})$
  $t \leftarrow t + 1$
fim enquanto

prática existem diversas instanciações populares, com maiores ou menores divergências em relação ao arquétipo algorítmico apresentado. A escolha do algoritmo para aplicação a um problema específico pode ser feita atendendo a factores que vão desde a melhor adequação de uma abordagem ao domínio do problema até à preferência pessoal do utilizador, passando pela possibilidade de aplicação directa do tipo de representação utilizada. Nas próximas secções procuramos, de forma breve, descrever as abordagens mais importantes.

3.3 Abordagens Clássicas

Nesta secção apresentamos as abordagens que consideramos terem maior importância histórica. É sobretudo devido a esta importância, resultante não só de estarem entre os primeiros algoritmos evolucionários apresentados na bibliografia, mas também da influência que tiverem em muitas outras abordagens que surgiram posteriormente, que podem ser consideradas as abordagens clássicas da computação evolucionária. Note-se que esta denominação não implica de forma alguma uma menor utilização contemporânea, já que pelo menos duas delas continuam a ser extremamente populares.

3.3.1 Algoritmos genéticos

Proposto inicialmente por Holland [1975, 1992] e desenvolvido, entre outros, por Goldberg [1989], o algoritmo genético[3] (AG) constitui provavelmente a instanciação mais conhecida e aplicada do algoritmo evolucionário genérico, inclusivamente com ambas as denominações a serem frequentemente utilizadas de forma intermutável. Na sua formulação clássica, o algoritmo genético utiliza uma representação baseada em vectores

[3]Do inglês *genetic alghorithm*.

binários para os indivíduos. Esta permite, com maior ou menor esforço, codificar soluções candidatas para qualquer problema, havendo uma distinção clara entre o genótipo de um indivíduo (codificação binária) e o seu fenótipo (solução descodificada).

A variação é produzida através de dois operadores, um de recombinação e outro de mutação. A recombinação é denominada recombinação de um ponto e age sobre dois progenitores produzindo dois descendentes através da troca de material genético a partir de um ponto do genótipo escolhido aleatoriamente. A mutação altera o valor de um *bit*, sendo aplicada a cada posição do genótipo com uma probabilidade muito baixa. Tradicionalmente, a selecção era feita em proporção ao desempenho, sendo a totalidade da população (de tamanho fixo) substituída pela nova geração de descendentes.

O facto desta ser uma formulação conceptualmente simples, aliada à sua flexibilidade de utilização e facilidade de implementação, levou à utilização generalizada dos algoritmos genéticos nos mais diversos domínios. No entanto, a utilização da representação binária implicava um procedimento de descodificação frequentemente oneroso do ponto de vista computacional. Adicionalmente, tornava difícil a definição de operadores especializados que codificassem informação sobre o domínio para acelerar o processo de procura. Estas dificuldades motivaram muitos utilizadores destes algoritmos ao desenvolvimento de representações mais próximas do domínio das soluções, bem como à criação de operadores específicos para os problemas em causa.

Na utilização contemporânea dos algoritmos genéticos, a representação (bem como obviamente a função de desempenho) é escolhida de maneira a adequar-se o melhor possível ao domínio das soluções. Embora existam operadores padrão definidos para muitos domínios (vectores binários, inteiros, reais, permutações, listas, árvores, etc...) é frequentemente necessário algum nível de adaptação destes ao domínio, para garantir um melhor desempenho. Mitchell [1996] apresenta um bom texto introdutório aos algoritmos genéticos e Haupt and Haupt [2004] descrevem um conjunto significativo de aplicações.

3.3.2 Estratégias evolutivas

As estratégias evolutivas[4] (EE), embora tenham surgido mais cedo, tiveram o seu desenvolvimento inicial na década de 70 na Alemanha com os trabalhos de Rechenberg [1973] e Schwefel [1974, 1981]. Embora o seu estudo estivesse inicialmente mais centrado nos grupos alemães, rapidamente se tornaram algoritmos evolucionários de uso global [Beyer and Schwefel, 2002].

As EE têm tido um desenvolvimento paralelo com o dos algoritmos genéticos, embora provavelmente constituam uma abordagem evolucionária menos difundida. Um factor

[4]Do alemão *evolutionsstrategie*.

importante para esta menor difusão será, porventura, a sua natureza menos flexível. As estratégias evolutivas surgiram como um algoritmo de optimização paramétrica, assentando numa representação constituída por um par de vectores de valores reais $v_i = (\mathbf{x}_i, \sigma_i)$ para cada indivíduo i. O vector $\mathbf{x}_i$ contém os valores dos parâmetros a optimizar, enquanto σ_i codifica valores do desvio padrão associado a cada parâmetro. Este segundo vector tem um papel importante em termos da mutação, a qual constitui o operador de variação dominante neste algoritmo.

A mutação é realizada em duas fases, primeiro perturbando aleatoriamente o vector de desvios padrão σ_i e, de seguida, adicionando a cada posição do vector $\mathbf{x}_i$ um valor aleatório amostrado de uma distribuição normal com média 0 e desvio padrão retirado da mesma posição do vector de desvios padrões. Quando a recombinação é utilizada pode assumir duas formas: a recombinação discreta cria um descendente escolhendo aleatoriamente pares parâmetro/desvio padrão da mesma posição dos progenitores; a recombinação linear produz descendentes que são combinações lineares dos progenitores.

O mecanismo de selecção é específico das estratégias evolutivas, no sentido em que é aplicado após os operadores de variação e consiste basicamente em substituir a população inicial com os melhores indivíduos resultantes do processo de reprodução. Assume-se que a população inicial tem μ indivíduos e que, através dos operadores de variação, são produzidos λ descendentes. Se a nova população for escolhida apenas de entre os melhores λ descendentes, desprezando os μ progenitores, temos uma estratégia evolutiva do tipo (μ, λ)-ES, em que $\lambda > \mu$ para haver pressão selectiva. Caso a população seja escolhida de entre a totalidade dos indivíduos, incluindo os μ progenitores e os λ filhos, então é uma estratégia do tipo $(\mu + \lambda)$-ES.

A primeira estratégia evolutiva era do tipo $(1 + 1)$-ES, implicando um mecanismo simples de procura progenitor-descendente. A mutação era o único operador utilizado, sendo que o vector de desvios padrão σ_i se mantinha constante ou era alterado de forma determinista. A partir do progenitor era criado um descendente por mutação, o qual substituía o pai caso tivesse um melhor desempenho. Uma variação óbvia a este método é a estratégia $(1+\lambda)$-ES, na qual λ descendentes eram criados, sendo o melhor escolhido para progenitor da próxima geração, mais uma vez no caso de ser superior ao pai em termos de desempenho. Estas versões iniciais realizavam uma espécie de procura trepa-colinas estocástica, e, apesar do tamanho dos saltos ser adaptável através da evolução dos vectores σ, apresentavam as dificuldades comuns a este tipo de métodos no que diz respeito a lidar com múltiplos óptimos locais.

Para lidar com essa limitação foram introduzidas várias variantes do tipo (μ, λ)-ES (mais recomendada para problemas de optimização em $\mathbb{R}^m$) e $(\mu + \lambda)$-ES (indicada para problemas de optimização combinatória), as quais incluíam populações de μ progenitores e operadores de recombinação. A notação utilizada nas estratégias evolutivas

contemporâneas é $(\mu/\rho+, \lambda)$-ES, onde μ e λ têm os significados já conhecidos, enquanto ρ representa o número de progenitores envolvidos no operador de recombinação. Isto é, o número de progenitores de cada descendente, sendo que $\rho = 1$ indica clonagem e que apenas a mutação é usada. A utilização mutuamente exclusiva de + ou , indica respectivamente que a nova população são os melhores μ indivíduos escolhidos de entre os $\mu + \lambda$ progenitores e descendentes ou apenas de entre os λ descendentes.

A juntar à representação e mecanismos de selecção específicos, convém sublinhar um terceiro elemento que diferencia as estratégias evolutivas dos restantes algoritmos evolucionários. Já mencionámos a inclusão de um vector de desvios padrão σ_i na representação e a sua utilização no operador de mutação. O facto do próprio σ_i ser evoluído ao longo do algoritmo, torna as estratégias evolutivas nos únicos algoritmos evolucionários que, desde a sua concepção inicial, podem adaptar o seu mecanismo de procura, nomeadamente os operadores de variação, ao problema abordado, ao longo da execução do algoritmo. O facto dos desvios padrão poderem variar de parâmetro para parâmetro num mesmo indivíduo - e entre indivíduos - permite evoluir desvios padrões que, em vez de levarem a mutação a produzir descendentes numa hiper-esfera em torno da população, alongue a procura ao longo de eixos que produzam um maior incremento de desempenho nos indivíduos.

Na versão estado da arte das estratégias evolutivas para domínios reais, denominada Estratégia Evolutiva com Adaptação da Matriz de Co-Variância[5] (EE-AMC)[Hansen and Kern, 2004], é ainda dado um passo adicional, ao acrescentar à representação uma matriz $\mathbf{M}$, a qual armazena valores de co-variância entre os parâmetros. Esta adição permite alongar a área de geração de novos indivíduos em qualquer direcção no espaço de procura que produza os maiores incrementos no desempenho, à medida que $\mathbf{M}$ é ela própria evoluída pela estratégia evolutiva. Esta capacidade auto-adaptativa do mecanismo de procura é provavelmente responsável pelo facto da EE-AMC ser um dos algoritmos evolucionários com melhores resultados para optimização em $\mathbb{R}^m$. Note-se, no entanto, que a necessidade de armazenar uma matriz de dimensão $m(m-1)/2$ por indivíduo, implica dificuldades computacionais óbvias na aplicação desta abordagem a problemas de elevada dimensionalidade.

3.3.3 Programação evolucionária

Historicamente convém ainda mencionar a programação evolucionária[6] [Fogel, 1999; Fogel et al, 1966], já que, em conjunto com os algoritmos genéticos e as estratégias evolutivas, foi das primeiras abordagens do domínio da computação evolucionária a ser introduzida. Inicialmente tinha como objectivo produzir inteligência artificial através

[5] Do inglês *covariance matrix adaptation evolution strategy (CMA-ES).*

[6] Do inglês *evolutionary programming.*

da evolução de decisores baseados em estruturas de dados, por exemplo autómatos finitos, através de um algoritmo evolucionário centrado no operador de mutação. Quando aplicada a representações reais, a programação evolucionária é bastante semelhante às estratégias evolutivas, no sentido em que também se baseia na adição de perturbações gaussianas de média 0 aos valores reais de cada indivíduo. Outra semelhança inclui a utilização de mecanismos adaptativos para a escolha ou definição da mutação a utilizar.

3.4 Outros Algoritmos Evolucionários

Além das abordagens clássicas, existem vários outros algoritmos evolucionários que, embora de utilização menos difundida, são mesmo assim importantes, sobretudo por permitirem resultados competitivos em determinados domínios específicos de aplicação. Nesta secção apresentamos apenas três dos muitos existentes, dois escolhidos por terem relevância em várias discussões apresentadas nesta dissertação e o terceiro por servir de base a dois algoritmos que utilizámos no trabalho experimental.

3.4.1 Programação genética

Entre as adaptações do algoritmo genético a domínios específicos, a programação genética[7] (PG), desenvolvida por Koza [1994, 1992, 2003] para permitir a evolução de programas de computador, atingiu um tal nível de popularidade que merece ser considerada um algoritmo evolucionário distinto por mérito próprio [Banzhaf et al, 1997; Langdon and Poli, 2002]. A diferença essencial em relação aos algoritmos genéticos decorre da especialização da programação genética para trabalhar sobre uma representação baseada em árvores sintácticas. Estas árvores representam geralmente programas de computador ou algoritmos, os quais são executados ou interpretados para resolver ou construir uma solução para o problema que está a ser abordado.

As árvores utilizadas na programação genética são construídas a partir de dois conjuntos de nós: o conjunto de funções e o conjunto de terminais. O primeiro inclui as funções que podem ser utilizadas para resolver determinado problema e que constituem os nós internos de cada árvore, com um número de ramos igual à aridade da função. O conjunto de terminais inclui as constantes e variáveis admissíveis na representação, as quais são utilizadas como folhas nas árvores, já que não necessitam de argumentos.

O operador de recombinação implica a troca de sub-árvores entre dois progenitores, sendo estas escolhidas pela selecção aleatória do seu nó inicial. A sub-árvore a trocar englobará todos os nós e folhas que se seguem a esse nó. A mutação consiste na

[7] Do inglês *genetic programming*.

substituição de uma sub-árvore por outra gerada aleatoriamente a partir dos mesmos conjuntos de funções e terminais. Cada função deve poder receber como argumento o retorno de qualquer outra função, bem como os tipos de dados a que pertencem os elementos do conjunto de terminais. Esta condição garante que, a partir de árvores sintacticamente correctas, os operadores de mutação e recombinação criarão também apenas novas árvores sintacticamente correctas.

O operador de recombinação é particularmente destrutivo na programação genética, razão pela qual é normalmente necessário, durante a reprodução, efectuar a clonagem de uma percentagem da população anterior para a nova população. Este mecanismo permite preservar alguns dos bons indivíduos já existentes, evitando quebras drásticas no desempenho médio da população. Também pela mesma razão a mutação é um operador relativamente secundário (e ausente nas primeiras formulações do algoritmo) na programação genética. Como a troca de sub-árvores na recombinação não é efectuada segundo alguma limitação em termos de posicionamento dos pontos de inserção ou estrutura das sub-árvores, torna-se sempre possível, através deste operador, introduzir qualquer função ou terminal ainda existente na população em qualquer posição de uma árvore. A mutação é assim apenas estritamente necessária quando uma função ou terminal se extingue totalmente dentro da população.

Apesar do sucesso desta abordagem e da sua particular aptidão para problemas onde as soluções candidatas devem ser codificadas como algoritmos ou estruturas de dados complexas como árvores (por exemplo, árvores de decisão), também existem algumas dificuldades que se colocam na sua utilização prática. A mais importante tem ver com o crescimento desmesurado (também conhecido por *bloating*) dos indivíduos que se verifica durante a execução do algoritmo [Poli, 2003]. Este é um problema comum em programação genética e, tratando-se de uma representação de tamanho variável, implica a necessidade de estabelecimento de limites máximos para o tamanho dos indivíduos. No entanto, o tamanho médio da população acaba sempre por se aproximar deste limite, causando não só problemas do ponto de vista do desempenho computacional - necessidade de avaliar muitos indivíduos de grande dimensão - mas também em termos de perda de diversidade na população. Poli et al [2008] tratam algumas destas dificuldades inerentes à aplicação prática da programação genética.

3.4.2 Evolução diferencial

Uma abordagem recente com resultados promissores em termos de optimização de funções reais multi-variável é denominada evolução diferencial[8] (ED) [Price et al, 2005; Storn and Price, 1997]. Trata-se de uma forma de optimização heurística populacional, como a generalidade dos algoritmos evolucionários, com um operador de variação es-

[8]Do inglês *differential evolution*.

pecífico que utiliza cada elemento x da população e três outros elementos distintos da população para criar um novo indivíduo y, utilizando uma fórmula matemática simples. Caso o desempenho de y, dado pelo valor da função a optimizar nesse ponto, seja melhor que o desempenho de x, este é substituído por y na população, implementando assim um mecanismo de selecção. A evolução diferencial é um algoritmo simples e de fácil implementação, apresentando resultados competitivos quando comparada com outras abordagens evolucionárias em problemas de optimização em $\mathbb{R}^m$ [Vesterstrom and Thomsen, 2004], embora os resultados pareçam algo dependentes de uma escolha adequada dos parâmetros do algoritmo.

3.4.3 Algoritmos meméticos

Além do sucesso individual de cada família de algoritmos evolucionários, a hibridização entre abordagens evolucionárias ou entre estas e métodos tradicionais tem aumentado o número de ferramentas disponíveis para a resolução de problemas complexos, com resultados frequentemente superiores aos obtidos por métodos não hibridizados [Grosan and Abraham, 2007]. Existem inúmeros exemplos de algoritmos evolucionários híbridos na bibliografia da área, especialmente no que diz respeito à optimização por enxames de partículas. Estes algoritmos podem ser encontrados hibridizados, por exemplo, com evolução diferencial [Das et al, 2008], optimização por colónias de formigas [Shelokar et al, 2007] e algoritmos genéticos [Kao and Zahara, 2008].

De grande interesse é também a hibridização de algoritmos evolucionários com algoritmos de procura local. Os métodos resultantes combinam um algoritmo evolucionário - ou, de forma genérica, qualquer algoritmo populacional - com mecanismos de aprendizagem ou procura local centrados no indivíduo. Potenciam assim a capacidade exploratória global dos algoritmos evolucionários com estratégias de exploração local mais eficientes, por exemplo, utilizando informação disponível sobre o gradiente para encontrar rapidamente óptimos locais a partir dos indivíduos da população. Estes indivíduos melhorados são posteriormente devolvidos ao algoritmo evolucionário e integrados na população, onde os operadores evolucionários lhe serão aplicados da mesma forma que aos restantes indivíduos. Embora estes algoritmos surjam na literatura com várias denominações, como algoritmos evolucionários Lamarckianos ou algoritmos culturais, optaremos pela denominação de algoritmos meméticos[9] [Moscato, 1989; Moscato and Cotta, 2003]. O termo meme foi introduzido por Dawkins [1990] e refere-se uma unidade básica de transmissão de cultura ou imitação.

Um algoritmo memético básico está descrito em 3.2 e difere do algoritmo evolucionário genérico por adicionar a escolha de uma sub-população **q** e proceder iterativamente ao melhoramento, utilizando um algoritmo de aprendizagem ou procura local, de cada

[9]Do inglês *memetic algorithm*.

elemento dessa sub-população, antes de o devolver à população principal. Algoritmos meméticos mais complexos codificam o próprio mecanismo de melhoramento - o meme - no genótipo do individuo, de maneira que o próprio mecanismo é também evoluído pelo algoritmo [Ong et al, 2006; Smith, 2007].

Algoritmo 3.2 Algoritmo memético.

```
t ← 0
p ← iniciar(p)
p ← avaliar(p)
enquanto não terminar(t, p) faz
  p′ ← seleccionar(p)
  p″ ← variar(p′)
  q ← escolher(p″)
  para i = 1 → tamanho(q) faz
    melhorar(q_i)
  fim para
  p ← substituir(p″, q, p)
  p ← avaliar(p)
  t ← t + 1
fim enquanto
```

3.5 Inteligência de Enxame

Os algoritmos de optimização baseados em inteligência de enxame surgem frequentemente associados aos algoritmos evolucionários e à área da computação evolucionária, embora possam, com maior precisão, ser integrados numa área distinta denominada inteligência de enxame [Bonabeau et al, 1999; Eberhart et al, 2001]. A inteligência de enxame junta-se aos algoritmos evolucionários numa área mais geral e abrangente, a chamada computação natural. A investigação nesta área debruça-se sobre a utilidade computacional de arquitecturas e algoritmos inspirados pela natureza, em especial na tentativa de resolver problemas que resistem a outros métodos mais tradicionais. Além dos algoritmos de enxame e dos algoritmos evolucionários, são vários os métodos incluídos nesta área de estudo, como a computação amorfa[10] [Coore, 2005] ou os sistemas imunitários artificiais[11] [Castro and Timmis, 2003].

A metáfora utilizada nos algoritmos de enxame é substancialmente diferente da utilizada nos algoritmos evolucionários, embora esta última denominação seja frequentemente utilizada para englobar ambos os grupos. Em vez da selecção natural, a inteli-

[10] Do inglês *amorphous computation*.
[11] Do inglês *artificial immune systems*.

gência de enxame procura inspiração nos comportamentos cooperativos de sociedades de insectos e outros animais. Existem assim algoritmos inspirados nos mecanismos subjacentes ao voo coordenado de bandos de aves [Kennedy and Eberhart, 1995], nos trilhos de transporte das colónias de formigas [Dorigo, 1992], nos comportamentos exploratórios de enxames de abelhas [Karaboga and Akay, 2009] e muitos outros.

Apesar das diferentes metáforas utilizadas, muitos dos algoritmos de enxame, como a optimização por enxames de partículas, que discutiremos na próxima secção, possuem mecanismos que podem ainda assim ser mapeados no algoritmo evolucionário genérico que apresentámos. A utilização de uma população de soluções é um desses elementos comuns, mas também podem ser encontrados mecanismos de variação, selecção e estratégias de substituição. Estes paralelismos justificam a inclusão dos algoritmos de enxame na área da computação evolucionária, bem como o facto - que também ocorre nesta dissertação - desses algoritmos serem por vezes denominados algoritmos evolucionários.

3.5.1 Optimização por enxames de partículas

O algoritmo básico de optimização por enxames de partículas[12] (OEP) foi introduzido por Kennedy and Eberhart [1995] e procurava de alguma forma reflectir as interacções sociais em bandos de pássaros em busca de comida. As influências básicas que regem a trajectória de um indivíduo no bando são, por um lado, as suas próprias crenças sobre o comportamento que lhe trará maior benefício e, por outro, o desejo de partilhar o comportamento do bando, seguindo um líder ou um vizinho. Kennedy and Eberhart [1995] transformaram estes princípios num mecanismo genérico de optimização, denominado optimização por enxames de partículas, no qual as aves e o bando são substituídos pelos termos mais genéricos "partícula" e "enxame", com paralelismo nos "indivíduos" e "população" dos algoritmos evolucionários.

Na optimização por enxames de partículas, cada um dos elementos do enxame é representado por três vectores de tamanho m, assumindo um problema de optimização em $\mathbb{R}^m$. Cada indivíduo i é assim representado por um vector $\mathbf{x}_i$ que armazena a sua actual posição no espaço de procura e um vector $\mathbf{p}_i$ que armazena a posição correspondente ao melhor desempenho do partícula até à data. Um terceiro vector $\mathbf{v}_i$ é utilizado para armazenar os termos de perturbação do vector de posição. A cada iteração do algoritmo, a posição corrente das partículas é avaliada como sendo uma solução para o problemas de optimização e, caso o desempenho actual seja o melhor que a partícula atingiu até à data, a posição actual $\mathbf{x}_i$ é armazenada em $\mathbf{p}_i$. O vector de velocidades $\mathbf{v}_i$ é de seguida calculado de acordo com a equação (3.1) e utilizado para actualizar as posições das partículas.

[12] Do inglês *particle swarm optimization*.

$$\mathbf{v}_i^{t+1} = \mathbf{v}_i^t + \mathbf{u}(0,\phi_1) \otimes (\mathbf{p}_i^t - \mathbf{x}_i^t) + \mathbf{u}(0,\phi_2) \otimes (\mathbf{p}_g^t - \mathbf{x}_i^t) \quad (3.1)$$

$$\mathbf{x}_i^{t+1} = \mathbf{x}_i^t + \mathbf{v}_i^{t+1} \quad (3.2)$$

Na equação (3.1), $\mathbf{p}_i^t - \mathbf{x}_i^t$ representa a distância entre a partícula e a sua melhor posição em interacções anteriores e $\mathbf{p}_g^t - \mathbf{x}_i^t$ representa a distância entre a partícula e a melhor posição encontrada pelo enxame (ou um subconjunto deste que constitui a vizinhança da partícula), armazenada em $\mathbf{p}_g^t$; $\mathbf{u}(0,\phi_1)$ e $\mathbf{u}(0,\phi_2)$ são vectores de números aleatórios com distribuições uniformes entre 0 e ϕ_1 e 0 e ϕ_2, respectivamente; $\otimes$ é simplesmente o produto dos vectores, componente a componente.

Este cálculo da velocidade assume, na optimização por enxames de partículas, o papel do operador de variação dos algoritmos evolucionários. Mais especificamente, pode ser visto como uma forma de recombinação, na medida em que a nova posição de uma partícula irá depender da sua posição anterior, da sua melhor posição anterior e da melhor posição do enxame. Na metáfora social do enxame, a partícula é simultaneamente guiada pelas suas próprias convicções sobre onde a solução para o problema se encontra e também pelas crenças colectivas do posicionamento dessa mesma solução. Na prática, a posição de cada partícula oscilará entre aquelas duas posições, $\mathbf{p}_i^t$ e $\mathbf{p}_g^t$, explorando assim as áreas mais promissoras do espaço de procura.

Nesta formulação básica do algoritmo do enxame não existe um mecanismo que corresponda ao operador de mutação. Quanto à selecção, embora ela não exista de forma específica, podemos identificar, no entanto, mecanismos que permitem controlar a velocidade com que o enxame converge para a melhor posição encontrada, atingindo efectivamente o mesmo efeito que o realizado pela pressão de selecção. O primeiro desses mecanismos tem a ver com a escolha dos valores de ϕ_1 e ϕ_2. Valores elevados provocam uma maior oscilação no enxame e maior dificuldade em convergir. Um valor mais elevado de ϕ_2 em relação a ϕ_1 implica uma preferência da partícula pelo máximo do enxame, aumentando assim a pressão de selecção e velocidade de convergência. Outra forma de controlar a velocidade de convergência implica a imposição de limites à própria velocidade do enxame. Na realidade, não havendo essa limitação, a formulação anterior rapidamente diverge, produzindo velocidades inaceitavelmente elevadas. Foram feitas várias abordagens à limitação da velocidade, incluindo a imposição de uma velocidade máxima V_{max} e a utilização de um factor de inércia, o qual diminui ao longo do tempo, associado à velocidade [Banks et al, 2007, 2008; Poli et al, 2007].

Na formulação padrão actual das fórmulas de actualização, resultantes do estudo teórico do algoritmo realizado por Clerc [2006], o controlo da velocidade de convergência é efectuado através da utilização de um factor χ, o qual multiplica por toda a fórmula de actualização da velocidade. A escolha adequada deste parâmetro, bem como dos parâmetros ϕ_1 e ϕ_2 garante a convergência do enxame, sem necessidade de restrições

da velocidade. Adicionalmente, o estudo realizado por Clerc [2006], fornece indicações para a escolha dos valores daqueles parâmetros, permitindo assim evitar a sua determinação empírica. A formulação canónica é a apresentada nas equações (3.3) a (3.6), tipicamente com $\phi = 4.1$ e $\phi_1 = \phi_2$, resultando num $\chi \approx 0.7298$.

$$\mathbf{v}_i^{t+1} = \chi(\mathbf{v}_i^t + \mathbf{u}(0, \phi_1) \oplus (\mathbf{p}_i^t - \mathbf{x}_i^t) + \mathbf{u}(0, \phi_2) \oplus (\mathbf{p}_g^t - \mathbf{x}_i^t)) \tag{3.3}$$

$$\mathbf{x}_i^{t+1} = \mathbf{x}_i^t + \mathbf{v}_i^t \tag{3.4}$$

$$\text{com } \phi = \phi_1 + \phi_2 > 4, \tag{3.5}$$

$$\text{e } \chi = \frac{2}{\phi - 2 + \sqrt{\phi^2 - 4\phi}} \tag{3.6}$$

Embora com esta formulação e escolha de parâmetros o enxame convirja garantidamente, é ainda assim comum, na implementação prática do algoritmo, limitar efectivamente a velocidade máxima V_{max} em cada dimensão ao valor máximo admissível para essa dimensão X_{max}. Note-se que a formulação padrão omite a utilização de qualquer mecanismo que explicitamente emule a função dos operadores de mutação nos restantes algoritmos evolucionários. Assim, à medida que o enxame converge, ficando concentrado em zonas progressivamente mais pequenas do espaço de procura, não é possível às partículas escaparem para novas zonas do espaço de procura. Esta dificuldade torna o algoritmo mais susceptível à estagnação em torno de óptimos locais, a não ser que mecanismos adicionais de introdução de diversidade sejam utilizados. Existem várias propostas de mecanismos capazes de cumprir esta função na bibliografia da área, sendo que, por exemplo, Andrews [2006] compara experimentalmente vários operadores de mutação utilizados em conjunção com o algoritmo de optimização por enxames de partículas.

3.5.2 Optimização por colónias de formigas

Para problemas de optimização combinatória, especialmente problemas que possam ser reduzidos à procura de um caminho óptimo num grafo, o uso de optimização por colónias de formigas[13] tem obtido bons resultados [Dorigo, 1992; Dorigo and Blum, 2005; Dorigo and Stützle, 2004]. Este algoritmo faz parte da área dos algoritmos de enxame e da computação natural, e, embora surja frequentemente associado à computação evolucionária, baseia-se numa metáfora completamente distinta. A optimização por colónias de formigas está dependente do mecanismo de estigmergia - a coordenação indirecta entre agentes ou acções [Dorigo et al, 2000; Garnier et al, 2007]. O princípio base deste mecanismo é que o traço deixado no ambiente por uma acção irá influenciar

[13] Do inglês *ant colony optimization*.

o desempenho de uma próxima acção, seja esta realizada pelo mesmo agente ou por um diferente.

Na natureza, a estigmergia é observável, por exemplo, na capacidade de uma colónia de formigas em encontrar consistentemente o caminho mais próximo entre a entrada da colónia e as fontes de comida. Para tal, cada agente (formiga), vai deixando por onde passa um trilho de feromona, um químico que outras formigas podem detectar e utilizar nas suas tomadas de decisão. As formigas, quando sujeitas à escolha de uma trajectória, tendem a preferir caminhos com maior concentração de feromonas. Quando existem vários caminhos disponíveis sem feromona depositada as formigas seleccionam-nos com a mesma probabilidade. Mas à medida que as formigas chegam à comida e retornam, as que acidentalmente escolheram o caminho mais curto, realizam o percurso em menos tempo, de maneira que a concentração de feromoma tende a aumentar mais rapidamente nesse mesmo caminho. As próximas formigas irão ter assim uma probabilidade ligeiramente maior de escolher o caminho mais curto e também elas regressarem mais rápido, incrementando, mais uma vez, a concentração de feromona nesse caminho. O repetir deste processo termina com praticamente todas as formigas a escolherem de forma consistente o caminho mais curto.

Este princípio, traduzido para a procura de caminhos em grafos, através da deposição de "feromona" nas arestas por agentes computacionais simples, permite assim encontrar soluções para problemas complexos de optimização combinatória, como, por exemplo, o problema do caixeiro viajante ou o problema de roteamento de veículos. É especialmente competitivo com outras abordagens baseadas em meta-heurísticas, como os algoritmos genéticos ou a procura tabu, em formulações dinâmicas destes problemas. Embora esta abordagem tenha sido originalmente formulada de forma a lidar com problemas de optimização combinatória, existem já formulações para optimização em domínios contínuos [Dréo and Siarry, 2002; Socha and Dorigo, 2008], inclusivamente em ambientes dinâmicos [Shelokar et al, 2007].

Capítulo 4

Computação Evolucionária e MVS

Neste capítulo apresentamos o estado da arte da aplicação de algoritmos evolucionários e de enxame ao melhoramento de diversos aspectos das máquinas de vectores de suporte. Ocasionalmente são também mencionadas aplicações a outros métodos de núcleo, quando a metodologia utilizada surge como relevante para as MVS. Este estado da arte está divido em três secções, as quais englobam quatro áreas maiores de aplicação dos AE. A primeira secção inclui a optimização paramétrica das MVS, nomeadamente do parâmetro C e dos parâmetros do núcleo escolhido, e a selecção dos atributos que efectivamente são tidos em conta pelo classificador. Embora sejam tarefas independentes, tendem a ser abordadas em conjunto na literatura da área, o que levou à sua inclusão na mesma secção. A segunda secção aborda a utilização de técnicas evolucionárias na construção da própria função de núcleo. Por fim, na terceira secção, tratamos as abordagens evolucionárias ao treino das MVS.

4.1 Optimização Paramétrica e Selecção de Características

4.1.1 Optimização paramétrica

A escolha dos parâmetros de uma MVS pode influenciar significativamente o seu desempenho. Os parâmetros em questão incluem, por um lado, o parâmetro C de regularização, utilizado na função objectivo para controlar o equilíbrio entre complexidade do modelo e número de erros, e, por outro lado, os parâmetros específicos da função de núcleo escolhida. Em muita da bibliografia da área a função de núcleo é a função de base radial gaussiana (FBR), a qual possui um único parâmetro γ (ou σ) para definir. A técnica padrão para a escolha destes parâmetros consiste na utilização de um algori-

tmo de grelha. Este algoritmo define uma grelha de pontos sobre o espaço de procura, os quais são utilizados para treinar a máquina de vectores de suporte. O ponto correspondente ao melhor desempenho é escolhido para fornecer os valores definitivos dos parâmetros ou, alternativamente, uma grelha mais fina pode ser criada em torno deste ponto promissor, caso o desempenho não seja considerado satisfatório. Este algoritmo é, no entanto, pouco eficiente computacionalmente e facilmente fica preso em óptimos locais. Abordagens mais sofisticadas implicam a utilização de métodos de descida de gradiente, os quais implicam condições de diferenciabilidade sobre a função de núcleo e a medida de erro utilizada, além de não lidarem de forma adequada com possíveis óptimos locais.

Os algoritmos evolucionários são algoritmos de optimização global que facilmente podem ser aplicados a qualquer problema de optimização paramétrica. No contexto das máquinas de vectores de suporte, no caso mais simples, há apenas dois ou três parâmetros para optimizar, não sendo por isso inesperado que existam na bibliografia diversas abordagens à selecção dos parâmetros das MVS, utilizando diversas técnicas evolucionárias. Nos parágrafos seguintes revemos algumas das abordagens mais interessantes ou paradigmáticas da bibliografia.

Friedrichs and Igel [2005] utilizam EE-AMC, uma estratégia evolutiva, para optimizar os parâmetros da função de núcleo gaussiana, a qual é generalizada através da utilização de uma matriz de rotação e escala. São testados três modelos diferentes, optimizando apenas o parâmetro base, permitindo a escala independente dos diversos componentes do vector de entrada e permitindo simultaneamente escala e rotação dos dados. A codificação real representa os valores da matriz e ainda o parâmetro C, sendo por isso abordado um problema mais complexo do que a simples optimização de dois parâmetros. Uma investigação do desempenho obtido a partir dos parâmetros encontrados por um algoritmo de grelha permitiu observar que a optimização destes parâmetros pode ser um problema multimodal. Para vários conjuntos de dados de teste foi possível produzir melhorias em relação à optimização dos parâmetros base usando o algoritmo de grelha. Os melhores resultados foram obtidos pela abordagem que permite a escala independente dos diversos componentes do vector de entrada, sendo que a versão que permitia a rotação dos dados geralmente não produziu melhorias significativas em relação à versão anterior.

Também Runarsson and Sigurdsson [2004] utilizam estratégias evolutivas, mas apenas para evoluir o parâmetro C e o parâmetro σ de uma RBF. O aspecto interessante desta abordagem tem a ver com o facto de não utilizar validação cruzada ou um conjunto de teste para avaliar os indivíduos, antes utilizando vários critérios diferentes, com bases teóricas fortes, para estimar o desempenho em termos de erro de generalização das MVS resultantes. A medida de desempenho mais promissora parece ser $(R^2 + \frac{1}{C}) \sum \alpha_i$, sendo que os α_i são os parâmetros dos vectores de suporte e R é o raio da menor

hiper-esfera centrada na origem que engloba todos os dados de treino no espaço de características.

Pai [2006] utiliza um algoritmo genético com representação binária para optimizar os parâmetros C e ϵ de uma MVS utilizada para regressão numérica, bem como o parâmetro da função de núcleo, no contexto da previsão de fiabilidade e previsão de falhas. A solução final revelou-se mais precisa do que vários outros modelos de regressão utilizados, bem como em relação a MVS não optimizadas. Já Wu et al [2007] optimizam C e σ utilizando um algoritmo genético com representação real, num problema de classificação de insolvência. O classificador baseado na MVS optimizada pelo AG revelou-se mais preciso do que vários outros modelos, incluindo MVS simples, redes neuronais e análise discriminante.

Também a optimização por enxames de formigas tem sido utilizada na optimização paramétrica das MVS, o que não é inesperado, já que a sua representação original, utilizando valores reais, bem como a simplicidade de implementação, se prestam facilmente a este tipo de aplicação. Lee et al [2006] optimizam os parâmetros de uma MVS de mínimos quadrados para um problema de identificação de sistemas não lineares utilizando OEP. Guo et al [2008] fazem o mesmo, mas para diversos conjuntos de dados e comparando com outros classificadores. As MVS de mínimos quadrados, com os parâmetros optimizados, revelaram-se superiores em termos de precisão. Wang et al [2006] optimizam dois parâmetros da função de núcleo num problema de regressão numérica, utilizando máquinas de vectores de suporte de mínimos quadrados. Finalmente de Souza et al [2006] utilizam optimização por enxames de partículas para optimização dos parâmetros de MVS para problemas de classificação múltipla. Neste caso é necessário optimizar um par de parâmetros para cada MVS.

Hong et al [2007] usam uma forma diferente de inteligência de enxame, uma versão do algoritmo de optimização por colónias de formigas, para optimizar os parâmetros do núcleo e da MVS num conjunto de problema de regressão numérica relacionados com a previsão de valores cambiais. A MVS com optimização de parâmetros obteve melhores resultados experimentais que vários outros modelos de regressão, incluindo redes neuronais.

Alguns autores tentaram comparar diversas abordagens evolucionárias num mesmo problema, ou conjunto de problemas. Rossi and de Carvalho [2008] comparam várias abordagens evolucionárias na optimização de MVS com função de núcleo gaussiana. A comparação é feita utilizando quatro conjuntos de dados de expressão genética, sendo que os métodos comparados incluem optimização por colónias de formigas, algoritmos genéticos, optimização por enxames de partículas e um algoritmo de selecção clonal (outra forma de computação natural). Os resultados obtidos são também comparados com os valores sugeridos pelo pacote de software que implementa as MVS, o qual usa uma procura em grelha paralela para optimizar os parâmetros. O desempenho dos

indivíduos é medido, como habitualmente, através do erro de generalização das MVS estimado por validação cruzada. Os resultados não são claros, com diversas abordagens a apresentar vantagem em problemas diferentes. As abordagens evolucionárias foram sempre competitivas com a procura em grelha, sendo inclusivamente menos dispendiosas do ponto de vista computacional.

Gilsberts et al [2010] usam três algoritmos evolucionários diferentes para optimizar os parâmetros de uma formulação especial de MVS, as máquinas de vectores de suporte de mínimos quadrados. Os parâmetros são optimizados utilizando um AG com representação binária, outro com representação real e finalmente utilizando uma estratégia evolutiva. Um quarto modelo, que será discutido mais tarde, combina a optimização evolucionária de parâmetros com a evolução de combinações de funções de núcleo utilizando programação genética. Ao contrário de abordagens anteriores, os resultados experimentais nos conjuntos de dados testados, embora competitivos com os de um algoritmo de grelha utilizado (e necessitando de menos recursos computacionais), não resultaram em classificadores com desempenho claramente melhor.

4.1.2 Selecção de características

Uma dificuldade comum nos problemas de classificação consiste na escolha do subconjunto das características que mais contribuem para a determinação da classe, especialmente quando os dados de entrada possuem uma dimensionalidade extremamente elevada, com implicações óbvias em termos do custo computacional da construção do classificador. Esta dificuldade é ainda mais significativa quando os dados são muito ruidosos ou existe redundância significativa entre características. Uma área em que esta questão é particularmente relevante é na classificação de células ou tecidos a partir de dados de expressão dos genes (expressão genética). Nestes problemas a selecção de características é essencial, já que normalmente existem milhares de genes disponíveis na expressão genética de determinada amostra, enquanto apenas algumas dezenas são discriminantes em termos de classificação. Esta questão é comum a muitos outros problemas de biologia computacional. Statnikov et al [2005] apresentam resultados experimentais que permitem não só verificar que as máquinas de vectores de suporte são a abordagem com melhor desempenho neste tipo de problemas, mas ainda que esse desempenho pode ser significativamente melhorado através de uma selecção adequada das características utilizadas.

Existem duas abordagens básicas ao problema de selecção de características. Na primeira, a selecção é executada numa etapa de pré-processamento, independentemente do algoritmo de classificação, agindo assim como um filtro entre os dados originais e o algoritmo. Neste caso a informação sobre a precisão do classificador não está disponível e o filtro tem de utilizar outras medidas para avaliar a qualidade do subconjunto de

características. A segunda abordagem é denominada do tipo *wrapper*, ou envolvente, já que o algoritmo de classificação é envolvido pelo método de selecção. Cada subconjunto de características proposto é usado para treinar o algoritmo, sendo avaliado por uma estimativa do erro de generalização do classificador, geralmente através de validação cruzada.

A extracção de características utilizando algoritmos genéticos é comum quando feita em conjunto com outros classificadores. Por exemplo, Yang and Honavar [1998] usam um algoritmo genético para seleccionar as características fornecidas a uma rede neuronal e Raymer et al [2000] seleccionam características utilizando uma representação binária, enquanto simultaneamente evoluem pesos para as características seleccionadas. A classificação é feita utilizando um algoritmo do tipo k vizinhos mais próximos. Ambas as abordagens produziram resultados experimentais positivos, resultando num aumento da precisão da classificação com os subconjuntos de características seleccionadas. Em anos mais recentes, com a popularização das máquinas de vectores de suporte, têm surgido diversas abordagens que procuram aplicar algoritmos genéticos e outras técnicas evolucionárias na selecção de características para estes métodos.

A generalidade das abordagens evolucionárias à selecção de características opta por simultaneamente optimizar os parâmetros do núcleo e da máquina de vectores de suporte. Desta forma, o subconjunto das características e o modelo da MVS é optimizado conjuntamente, o que, além de resultar num problema mais interessante, tem a potencialidade de produzir melhores resultados experimentais. No entanto, alguns autores optam por seleccionar apenas as características. Peng et al [2003] usam um algoritmo genético com representação binária para seleccionar o conjunto de genes efectivos num problema classificação de tumores com múltiplas classes, sendo o classificador baseado numa formulação multi-classe das máquinas de vector de suporte. Huerta et al [2006] combinam algoritmos genéticos com lógica *fuzzy* para melhorar ainda mais a precisão de selecção de características, também na área da expressão genética.

Shen et al [2007] utilizam uma versão binária modificada do algoritmo de optimização por enxames de partículas para seleccionar características, não fazendo optimização de parâmetros da MVS. Neste caso o algoritmo é aplicado a conjuntos de dados de expressão genética com milhares de genes, melhorando significativamente o desempenho da MVS. Huang and Dun [2008] reforçam a viabilidade da versão binária padrão de OEP, utilizando conjuntos de dados sintéticos e uma arquitectura distribuída para aliviar a carga computacional deste tipo de abordagem.

As abordagens seguintes, que tratam simultaneamente características e parâmetros, serão provavelmente as mais interessantes desta secção. Jack [2002] apresentou uma das primeiras abordagens evolucionárias à selecção de características para máquinas de vectores de suporte, evoluindo simultaneamente um parâmetro da função de núcleo. No entanto, não optimizou o parâmetro C da função objectivo. Usou um algoritmo

genético padrão numa aplicação de detecção de falhas com máquinas de vectores de suporte e redes neuronais multi-camada. O algoritmo genético conseguiu encontrar características e um valor para o parâmetro de núcleo que melhoraram o desempenho de ambos os classificadores, tendo a rede neuronal mantido os melhores resultados ao longo das experiências.

Numa abordagem mais bem sucedida, Frohlich et al [2003] usam uma variante do algoritmo genético tradicional para evoluir os subconjuntos de características, utilizando uma representação binária quando o número de características desejadas é desconhecido *a priori* e uma representação decimal, mais parcimoniosa, no caso desse número estar predeterminado. Na representação binária, cada bit representa a selecção ou não da característica correspondente. Na representação decimal, cada um de n inteiros representa o índice de uma característica seleccionada.

Trata-se de uma abordagem envolvente, já que uma MVS é treinada com o conjunto de características correspondente a cada indivíduo da população. Uma das vantagens da utilização de MVS para classificação - o facto de exigirem limites máximos teóricos para o erro de classificação - é utilizada por Frohlich et al [2003] para evitar a utilização da validação cruzada na avaliação do classificador. Esta é substituída pelo cálculo de um desses limites para cada MVS, com ganhos em termos de custo computacional. Frohlich et al [2003] foram também dos primeiros autores a implementar a evolução do parâmetro de regularização C em simultâneo com o conjunto de características, reconhecendo que a escolha de um diferente conjunto de características poderia influenciar o valor óptimo de C. A optimização evolucionária dos parâmetros de núcleo é sugerida mas não implementada.

Os algoritmos genéticos foram testados em várias configurações com dois conjuntos de dados sintéticos e dois conjuntos de dados constituídos por micro-arrays de DNA, usando tanto a validação cruzada como os referidos limites de erro das MVS. Para comparação foram utilizadas duas abordagens padrão do tipo filtro e uma do tipo envolvente. Verificou-se que a abordagem evolucionária utilizando o limite de erro era competitiva com as abordagens pré-existentes, especialmente quando o número de características desejadas não era definido à partida.

Nguyen et al [2004] utilizam um algoritmo genético para encontrar simultaneamente as características a considerar e uma combinação de núcleos que resulte numa elevada precisão classificativa da máquina de vectores de suporte, avaliada por validação cruzada. A representação utilizada é parcialmente binária (para as n características e $m-1$ operadores de ligação entre núcleos), inteira (m inteiros para os expoentes de m núcleos) e real ($2 \times m$ parâmetros para os núcleos). Embora os resultados experimentais sejam limitados a um único conjunto de dados e as possibilidades de combinação de núcleos estejam limitadas a três núcleos combinados apenas por adição ou multiplicação, esta é uma das primeiras abordagens que procura encontrar evolucionariamente

um núcleo mais adequado a determinado problema, sendo este aspecto discutido mais detalhadamente na secção seguinte.

Ahn et al [2006] acrescentam um elemento adicional ao tratamento dos dados, ao incluir numa representação binária, a ser evoluída por um AG, n bits correspondentes às n instâncias, determinando assim, também de forma evolucionária, quais as instâncias consideradas pelo algoritmo de treino. Além das instâncias, também as características são seleccionadas evolucionariamente. Os parâmetros do núcleo (função de base radial gaussiana) são codificados de forma binária e também optimizados pelo algoritmo genético. Os resultados experimentais são promissores, embora sejam pouco significativos já que apenas é utilizado um conjunto de dados.

Huang and Wang [2006] fazem a selecção de características e, simultaneamente, optimizam os parâmetros da MVS (C e σ) utilizando um algoritmo genético com representação binária. A função de avaliação incorpora desempenho preditivo, número de características e ainda custos específicos para cada característica. Este aspecto multiobjetivo da função de avaliação distingue esta abordagem das anteriores. O algoritmo foi testado em vários conjuntos de dados, com características na ordem das centenas, e demonstrou obter maior precisão e utilizar menor números de características do que um algoritmo de grelha usado para comparação. Huang and Chang [2007] também usam um algoritmo genético na selecção de características e optimização de parâmetros, mas neste caso o AG utilizado é uma variante denominada algoritmo genético inteligente, com uma representação definida propositadamente para esta aplicação (embora ainda binária) e utilizando um operador de recombinação específico para acelerar a procura. O algoritmo é utilizado com resultados muito positivos em vários conjuntos de dados de expressão genética.

Note-se que os trabalhos descritos até aqui são todos baseados em algoritmos genéticos. Consideramos que este facto se deve à naturalidade da utilização de uma representação binária, de aplicação directa no caso da selecção de características (ou mesmo das instâncias), que é a representação tradicionalmente utilizada nos AG. A escolha deste algoritmo surge assim como natural nas primeiras abordagens evolucionárias, já que não é necessário desenvolver novas representações nem operadores, podendo o algoritmo ser utilizado de forma directa. Inclusivamente, nota-se que os parâmetros são também frequentemente codificados de forma binária, de maneira a manter a simplicidade da abordagem. Com a proposta de versões binárias do algoritmo de optimização por enxames de partículas [Kennedy and Eberhart, 1997], também rapidamente começaram a surgir abordagens baseadas em OEP para o problema de selecção de características, embora alguns autores tenham preferido utilizar a representação real do algoritmo básico.

Lin et al [2008] usam um algoritmo de optimização por enxames de partículas básico, com representação real, na optimização dos dois parâmetros da máquina de vectores

de suporte e na selecção de características. Para tal, uma característica é seleccionada quando o respectivo valor no vector $\mathbf{x}$ é superior a 0.5, variando aquele entre 0 e 1. Esta é possivelmente a implementação mais directa utilizado OEP e os resultados, numa variedade significativa de conjuntos de dados (embora todos com menos de 2000 características), são superiores, tanto aos obtidos usando um algoritmo de grelha para a escolha dos parâmetros, como aos presentes em Huang and Wang [2006], que aplicam um algoritmo genético nos mesmos conjuntos de dados.

Melgani and Bazi [2008] abordam a classificação de sinais de electrocardiograma, utilizando MVS e optimização por enxames de partículas na selecção de características e optimização de parâmetros. No contexto deste problema, é feita uma exploração experimental exaustiva em termos do desempenho das máquinas de vector de suporte, utilizando tanto análise de componentes principais como OEP, havendo ainda comparações com um algoritmo de k vizinhos mais próximos. A representação utilizada é a clássica (valores reais), tal como a versão do algoritmo de optimização por enxames de partículas. Interessantemente, a avaliação das partículas é feita utilizando o números de vectores de suporte como medida do limite do erro esperado para a MVS correspondente. Os melhores resultados foram obtidos pelas MVS combinadas com OEP.

Alguns trabalhos comparam várias abordagens evolucionárias para os mesmos conjuntos de dados. Samanta and Nataraj [2009] comparam os resultados obtidos por um algoritmo de OEP binário com a versão padrão, utilizando a representação real, na selecção de características e optimização de parâmetros para um problema de detecção de falhas. O autor não encontrou diferença significativa entre os resultados das diferentes representações. Yuan and Chu [2007] utilizam uma versão binária modificada do algoritmo de optimização por enxames de partículas para seleccionar características e optimizar os parâmetros da MVS no contexto de um problema de detecção de falhas. Os resultados experimentais obtidos são particularmente interessantes, já que permitem comparar entre o desempenho obtido pela abordagem que utiliza OEP, uma implementação que utiliza um AG binário e ainda uma terceira abordagem utilizando OEP na optimização dos parâmetros e análise de componentes principais (uma técnica de redução de dimensionalidade dos dados) para a selecção de características. A abordagem baseada em optimização por algoritmos de enxames revelou-se superior às restantes. Esta conclusão é, no entanto, restrita ao âmbito de um problema específico.

4.1.3 Conclusões

A optimização de parâmetros nas máquinas de vectores de suporte e selecção de características efectivamente utilizadas foram ambas abordadas por diversos autores, utilizando técnicas provenientes das várias áreas da computação evolucionária. Não sendo

ainda problemas fechados, existe, no entanto, um corpo significativo de investigação sobre o qual podemos retirar algumas conclusões:

- A optimização isolada dos parâmetros das funções de núcleo e das máquinas de vectores de suporte é possivelmente o problema mais tratado e ao qual foi aplicada a maior variedade de algoritmos evolucionários. Os resultados são geralmente competitivos com as abordagens tradicionais (algoritmos de grelha, descida do gradiente), resultando frequentemente num esforço computacional menor ou em melhor precisão do classificador. Para a optimização de apenas dois parâmetros, parece-nos discutível que, no caso geral, seja realmente necessário ir além dos valores sugeridos pelo pacote de treino de MVS que é utilizado.

- No entanto a optimização dos parâmetros associado à selecção de características revela-se um mecanismo poderoso para melhoria do desempenho nas máquinas de vectores de suporte. A selecção de características sempre foi tida como essencial, sobretudo em problemas como os de expressão genética, onde existem milhares de característica diferentes. Também já anteriormente os algoritmos evolucionários haviam demonstrado ser particularmente adequados a essa tarefa, em combinação com outros classificadores. No entanto, a utilização de algoritmos evolucionários associados a máquinas de vectores de suporte é particularmente interessante, já que permite, simultaneamente, seleccionar as características e os parâmetros do núcleo que resultam no melhor desempenho. A realização destas tarefas de forma independente pode não conseguir os mesmos resultados. Tratando-de um problema complexo, parcialmente de optimização combinatória e parcialmente de optimização real, é particularmente adequado à utilização de computação evolucionária.

- Nos artigos referidos podem ser encontradas abordagens realizadas com vários algoritmos evolucionários, incluindo algoritmos genéticos, estratégias evolucionárias, optimização por colónias de formigas e optimização por enxames de partículas. Entre os algoritmos mais representativos nota-se apenas a possível ausência da evolução diferencial e falta de abordagens meméticas, i.e., combinando técnicas de aprendizagem ou optimização local com os algoritmos de optimização global. Como seria de esperar num conjunto de abordagens tão diversas, não existe um algoritmo evolucionário que se revele claramente superior, embora algumas comparações pareçam apontar para alguma vantagem dos algoritmos de optimização por enxames de partículas, sobretudo quando se tem em atenção a complexidade de implementação de cada método.

- Do ponto de vista das representações utilizadas parece haver vantagem na utilização de representações directas, reais para os parâmetros e binárias para a selecção de características. Estas permitem implementações mais directas dos algoritmos,

sem necessidade de tradução das representações, não sendo detectável na bibliografia uma influência significativa da representação utilizada na qualidade dos classificadores finais. Esta constatação pode ser determinante para a escolha do algoritmo evolucionário utilizado. Na prática, especialmente no problema mais relevante - optimização dos parâmetros e selecção simultânea das características - pode ser mais simples e eficiente escolher algoritmos capazes de lidar com ambas as representações em simultâneo sem necessidade de mudanças significativas no algoritmo. Em especial os algoritmos de optimização por enxames de partículas podem ser utilizados quase sem modificações em ambas as representações.

- Um aspecto particularmente interessante da combinação de algoritmos evolucionários com máquinas de vectores de suporte prende-se com a avaliação dos indivíduos, as máquinas de vectores de suporte já treinadas. Quando utilizados com outros classificadores, os algoritmos evolucionários dependem geralmente de uma forma de validação cruzada para avaliar o classificador. A validação cruzada divide o conjunto de dados disponível em n subconjuntos (tipicamente 10), utilizando consecutivamente cada subconjunto como conjunto de teste, enquanto a união dos restantes é utilizada como conjunto de treino. O desempenho final é igual à média dos desempenhos dos n classificadores construídos. Consegue assim uma boa estimativa do desempenho do classificador em dados desconhecidos, aproveitando de forma eficiente os dados disponíveis, mesmo quando são escassos. Este aproveitamento tem no entanto um custo computacional acrescido, já que implica n execuções do algoritmo de treino. No caso das MVS, no entanto, alguns autores (como Frohlich et al [2003]) exploram uma das suas maiores vantagens, a existência de limites teóricos máximos para o erro de generalização, utilizando uma dessas medidas como estimativa do desempenho dos indivíduos evoluídos pelo algoritmo. Aceleram assim significativamente (por um factor de 10 em relação à validação cruzada com 10 subconjuntos) a execução do processo evolutivo.

4.2 Evolução de Funções de Núcleo

A escolha da função de núcleo utilizando técnicas evolucionárias acaba por ser uma sequência natural da utilização dos mesmos algoritmos para optimizar o núcleo através da procura dos parâmetros deste que levem ao melhor desempenho do classificador. Com a consciência de que o desempenho duma MVS depende fortemente tanto do núcleo escolhido como dos seus parâmetros, e de que a escolha do núcleo mais adequado a um problema específico é basicamente um processo de tentativa e erro, várias linhas de investigação procuraram automatizar esse processo de escolha utilizando algoritmos evolucionários. Essas linhas dividem-se basicamente em duas linhas de abordagem.

Alguns autores procuram utilizar algoritmos evolucionários variados para criar novos núcleos a partir de funções de núcleo já existentes. Outros procuram agir a um nível mais básico, utilizando programação genética para evoluir funções de núcleo completamente diferentes, a partir de funções matemáticas primitivas.

4.2.1 Combinação de núcleos

No contexto das abordagens que procuram usar núcleos bem conhecidos como elementos base dos novos núcleos, Phienthrakul and Kijsirikul [2005] utilizam uma estratégia evolutiva (5+10)-ES para construir funções de núcleos que são a soma pesada de n funções de base radial. Cada indivíduo codifica $2n$ parâmetros reais, correspondendo a dois parâmetros por FBR. Um dos parâmetros corresponde ao valor de σ e o outro ao peso da função de base radial a que está associado. Foram realizadas experiências com n variando entre 1 e 5, sendo que as combinações de 5 FBR apresentaram os melhores resultados na generalidade dos conjuntos de dados testados. A avaliação dos indivíduos foi feita utilizando validação cruzada. Note-se que esta abordagem garante que as funções de núcleo evoluídas são positivas semi-definidas, já que a soma pesada de núcleos positivos semi-definidos é ela própria um núcleo positivo semi-definido.

Nguyen et al [2004] também constroem novos núcleos por combinação de núcleos base, permitindo no entanto que a operação de combinação (adição ou multiplicação) entre cada par de operadores seja igualmente escolhida pelo algoritmo. A representação codifica um expoente e os parâmetros para cada núcleo, bem como as operações de combinação entre cada par. O algoritmo utilizado é um algoritmo genético e a avaliação é feita utilizando validação cruzada. As experiências realizadas sobre um único conjunto de dados permitiram concluir que a melhor combinação de núcleos apresentou resultados superiores aos obtidos por qualquer um dos núcleos (FBR, sigmóide e multiquadrático inverso) individualmente.

4.2.2 Síntese de novos núcleos

A definição de novos núcleos por combinação de núcleos primitivos tem no entanto limitações óbvias em termos de flexibilidade, já que se baseiam obrigatoriamente num conjunto fixo de funções de núcleo base. Em termos de computação evolucionária, a forma mais natural de ultrapassar esta limitação consiste na utilização de programação genética para evoluir o próprio código dos núcleos em vez dos parâmetros de cada núcleo base e/ou os operadores de combinação. A programação genética é extremamente flexível, permitindo combinar a evolução de núcleos completamente originais, a optimização dos seus parâmetros e mesmo a sua combinação com funções de núcleo tradicionais.

Howley and Madden [2005] apresentam uma das primeiras abordagens à utilização de programação genética para a síntese de novas funções de núcleo. Como conjunto de terminais utilizam dois vectores de dados x e y e as funções incluem versões escalares e vectoriais dos operadores $+$, $-$ e $\times$. O algoritmo de programação genético utilizado é relativamente padrão, com a excepção de serem utilizadas 5 populações em paralelo, sendo devolvido o melhor indivíduo encontrado em todas elas. Dada uma árvore t devolvida pelo algoritmo, o valor retornado pela função de núcleo para dois vectores x e y é o produto interno $\langle treeEval(t,x,y), treeEval(t,y,x)\rangle$, sendo $treeEval(x,y)$ o resultado da avaliação da árvore t recebendo os vectores x e y como terminais. Este mecanismo garante que a função de núcleo é simétrica, mas não garante que é positiva semi-definida. A escolha da função de núcleo é interessante: contrariamente ao habitual é utilizado o erro de treino como medida de desempenho, de maneira a evitar o custo computacional da validação cruzada. Para evitar o sobre-ajustamento resultante desta função de desempenho é utilizado o seguinte termo como factor de desempate: $\sum(\alpha_i) \times R^2$, com α_i os parâmetros dos vectores de suporte e R o raio da menor hiper-esfera centrada na origem que engloba todos os dados de treino no espaço de características. A ideia é privilegiar as MVS com uma margem maior em relação ao raio da esfera que engloba os dados, o que deverá privilegiar a sua capacidade de generalização.

Os resultados experimentais, embora obtidos para um conjunto relativamente pequeno de problemas, sugerem três conclusões principais:

- Em primeiro lugar, ao comparar os resultados obtidos por vários núcleos (FBR, sigmóide, polinomial) com variação de parâmetros, foram obtidas precisões de classificação muito diferentes, resultado que vinca a importância da escolha da função de núcleo bem como dos seus parâmetros.
- Apesar do selecção algo limitada de funções e terminais, a abordagem apresentada obteve maior precisão em vários dos conjuntos de treino e resultados competitivos nos restantes. A viabilidade da abordagem à síntese de novas funções de núcleo por programação genética fica assim demonstrada.
- O facto dos núcleos evoluídos não serem garantidamente positivos semi-definidos - logo a optimização converge para o primeiro óptimo local que encontra - sugere que com a utilização de métodos de treino globais poderiam ser obtidos ainda melhor resultados.

Diosan et al [2007] também usam uma abordagem semelhante à de Howley and Madden [2005], embora introduzam algumas alterações significativas:

- O conjunto de funções é expandido, englobando várias normas e operações entre vectores e separando funções escalares das vectoriais. Isto obriga a um maior

cuidado em termos de representação, já que é necessário garantir que as árvores geradas inicialmente, bem como as resultantes da aplicação dos operadores, são sintacticamente correctas.

- O conjunto de funções é escolhido de maneira a garantir que os núcleos resultantes são obrigatoriamente positivos semi-definidos.
- A avaliação é feita utilizando um sub-conjunto de dados reservado para o efeito, o qual não é portanto utilizado no treino da MVS. Note-se que esta abordagem só é funcional se houver um número relativamente elevado de instâncias disponíveis.

Os resultados experimentais em quatro conjuntos de dados demonstram que esta abordagem obtém resultados ligeiramente melhores do que os obtidos com funções de núcleo tradicionais, bem como do que os obtidos por Howley and Madden [2005] para os mesmos conjuntos de dados.

Girdea and Ciortuz [2007] usam uma ideia ligeiramente diferente das anteriores, recuperando a ideia de combinação de núcleos base, mas usando PG para a fazer. Para tal o conjunto de terminais é constituído pelos núcleos mais comuns (linear, polinomial e FBR), embora sendo adicionadas constantes geradas aleatóriamente. O conjunto de funções é $\{+, \times\}$. A avaliação é feita utilizando validação cruzada. Os resultados obtidos foram significativamente melhores do que os de uma MVS utilizando a FBR, mas, como foram utilizadas diversas estratégias de *boosting* durante o treino das MVS com núcleos evoluídos por PG, o efeito deverá ser repartido por ambos os aspectos da abordagem.

Sullivan and Luke [2007] utilizam uma metodologia semelhante à de Girdea and Ciortuz [2007], embora não incluindo os aspectos de *boosting* e expandindo o conjunto de combinações permitidas de maneira a incluir a multiplicação por uma constante e a função exponencial. As operações permitidas continuam a garantir que o núcleo resultante da sua aplicação aos núcleos base (neste caso polinomial, FBR e sigmóide) resultam num núcleo positivo semi-definido. As MVS com as funções de núcleo evoluídas foram comparadas com MVS utilizando a FBR e procura em grelha para definição dos parâmetros em seis conjuntos de dados, tendo obtido melhores resultados em termos de precisão de classificação em quatro deles. Note-se ainda a abordagem de Gilsberts et al [2010], semelhante às duas anteriores, mas que não apresenta diferenças tão significativas em termos de desempenho, facto parcialmente atribuído pelo autor às dificuldades em encontrar uma configuração óptima para o algoritmo de PG, especialmente com os custo computacionais inerentes a este algoritmo evolucionário.

Uma última abordagem à evolução de funções de núcleo digna de menção é apresentada por Gagné et al [2006]. Esta distingue-se das anteriores por não estar associada a máquinas de vectores de suporte, mas antes ao algoritmo do vizinho mais próximo,

o qual, sendo baseado numa medida de distância, pode também ser generalizado com a utilização de núcleos. Embora não use as máquinas de vectores de suporte, este método traz contribuições interessantes ao problema da evolução de funções de núcleo, sendo a mais relevante o facto de fazer a co-evolução dos conjuntos de treino e de teste simultaneamente com o núcleo propriamente dito. Efectivamente o algoritmo utiliza uma estratégia evolutiva para co-evoluir um conjunto de protótipos, o qual é um subconjunto dos dados de treino, evoluído de maneira a conduzir a núcleos de maior desempenho. Ou seja, a co-evolução entre núcleos e conjunto de protótipos é cooperativa. Já o conjunto de teste é evoluído, também utilizando uma estratégia evolutiva, mas agora de forma competitiva. Assim, um melhor conjunto de teste é aquele em que algoritmo tem mais dificuldade em classificar correctamente os exemplos que dele fazem parte. Esta ideia é particularmente interessante, já que permite atenuar o maior problema da evolução de núcleos usando programação genética - a sua complexidade computacional.

4.2.3 Conclusões

A escolha da função de núcleo mais adequada a um determinado problema é um problema não trivial que utilizadores menos experientes apenas poderão resolver por tentativa e erro. A computação evolucionária fornece ferramentas capazes de abordar este problema de forma automática, sendo possível apontar alguns resultados promissores da análise da ainda escassa bibliografia da área, bem como as limitações que lhe estão inerentes:

- A maior limitação, referenciada em vários dos artigos analisados, tem a ver com o custo computacional da combinação da programação genética com as máquinas de vectores de suporte. Tal como já tínhamos mencionado na secção sobre optimização de parâmetros e selecção de características, a utilização de validação cruzada em conjunto com algoritmos evolucionários exige recursos computacionais significativos. Este problema extrema-se aquando da utilização da programação genética, já que a necessidade de interpretar os indivíduos (árvores sintácticas) torna este algoritmo em particular ainda mais exigente computacionalmente. Vários trabalhos tentam minimizar este problema, sendo a solução mais comum a opção por medidas de desempenho baseadas em estimativas dos limites de erro das máquinas de vectores de suporte, abordagem cujas vantagens já discutimos na secção anterior. A ideia de co-evoluir conjuntos de treino e teste introduz uma possibilidade adicional em termos de diminuição das exigências computacionais.

- As abordagens encontradas limitam-se a problemas em que os dados se encontram representados sobre a forma de vectores reais, de forma que os núcleos evoluídos

trabalham basicamente nesses domínios. Não foram encontradas abordagens que procurem evoluir núcleos para dados mais estruturados, o que, atendendo à vasta área de aplicação actual das MVS, sugere muitas possibilidades de trabalho futuro, especialmente em áreas como a biologia computacional, onde a complexidade das estruturas tratadas implica a utilização de núcleos muito específicos.

- Apesar das limitações, todas as abordagens apresentaram resultados significativamente melhores, pelo menos em parte dos conjuntos de dados em que foram testadas, quando comparadas com funções de núcleo tradicionais. Estes resultados validam a propriedade da abordagem e sugerem que, atendendo ao número diminuto de abordagens descritas, e ao facto de ser uma área de investigação recente, melhores resultados serão obtidos à medida que mais trabalho for desenvolvido na área.

- Um aspecto central da evolução de funções de núcleo consiste na garantia dos núcleos gerados serem positivos semi-definidos, de maneira a que o problema de optimização da MVS tenha um único óptimo global. A tendência para a utilização da composição de núcleos base, como metodologia preferencial para a geração de novos núcleos, deve-se assim, provavelmente, ao facto de as operações de combinação utilizadas ($+$, $\times$, ...) serem escolhidas para garantirem que o núcleo composto se mantém positivo semi-definido. As abordagens que usam funções genéricas para geração dos núcleos são no entanto bastante mais flexíveis, especialmente se considerarmos a sua aplicação a domínios de dados mais diversificados.

- Mesmo nas abordagens que não garantem que os núcleos encontrados são positivos semi-definidos, as MVS treinadas continuam a ter resultados no mínimo competitivos. No entanto as técnicas tradicionais de treino das MVS não estão preparadas, como já mencionámos, para lidar com as funções multimodais resultantes de núcleos não positivos semi-definidos, parando, na melhor das hipóteses, no primeiro óptimo encontrado. Este óptimo pode ou não ser o óptimo global, podendo inclusivamente ser uma solução sub-óptima, significativamente pior que as restantes. Este facto abre perspectivas interessantes à combinação destes métodos com outras técnicas de treino de MVS, mais especificamente as baseadas em algoritmos evolucionários, já que a capacidade destes para procurar globalmente lhes permite lidar mais satisfatoriamente com problemas de optimização multimodais.

4.3 Treino de MVS Utilizando Computação Evolucionária

O treino de máquinas de vectores de suporte utilizando algoritmos evolucionários abre possibilidades que os métodos de treino tradicionais não permitem, nomeadamente a utilização de funções de núcleo não positivas semi-definidas e a optimização concorrente de vários objectivos. Permite ainda a possibilidade de trabalhar sobre qualquer uma das representações do problema de treino, tanto a primal como a dual. Finalmente, sendo os algoritmos de treino tradicionais de implementação algo complexa, o que limita o treino à utilização de pacotes de software pré-existentes, a utilização de algoritmos evolucionários, geralmente de mais simples implementação, permite que mais utilizadores de MVS implementem os seus próprios algoritmos de treino.

Apesar do interesse e possíveis aplicações do treino evolucionário de MVS encontram-se na bibliografia da área apenas três linhas de investigação, todas recentes, bem como um número igualmente diminuto de autores que as exploram. Nas próximas secções apresentamos de forma breve estas abordagens, dando ênfase às respectivas contribuições e limitações.

4.3.1 Optimização linear por enxames de partículas

A primeira abordagem que encontramos na bibliografia ao treino de máquinas de vectores de suporte surge em [Paquet and Engelbrecht, 2003b], onde é proposta a optimização dos parâmetros α do problema dual, utilizando o algoritmo de optimização por enxame de partículas. Esta abordagem parte dos métodos de decomposição geralmente utilizados para resolver o problema de programação quadrática associado ao treino de uma MVS. Estes métodos resolvem de forma numérica (com excepção da OSM) um subproblema correspondente a um subconjunto dos exemplos de treino, processo que é repetido iterativamente até o óptimo ser encontrado.

Partindo da abordagem de base, também aqui a OEP é utilizada para resolver subproblemas sucessivos e não o problema global. Para tal foi desenvolvida uma versão linear do algoritmo, chamada optimização linear por enxames de partículas (OLEP), a qual introduz alterações ao algoritmo de OEP básico no sentido de satisfazer a restrição linear do problema de optimização. A ideia básica é que, estando a função a optimizar sujeita a restrições lineares do tipo $\mathbf{Ap} = \mathbf{b}$, o enxame deve ser obrigado a "voar" no hiperplano $\mathbf{P}$ de soluções admissíveis, $\{\mathbf{p} \in \mathbf{P} | \mathbf{Ap} = \mathbf{b}\}$, consistindo portanto num método de preservação da admissibilidade das soluções.

O algoritmo garante tal comportamento ao iniciar o enxame de maneira a que a posição inicial $\mathbf{p}_i^0$ de cada partícula $\mathbf{p}_i$ obedeça a $\mathbf{Ap}_i^0 = \mathbf{b}$, i.e. se encontre no hiperplano de

soluções admissíveis. Alterando as equações originais do algoritmo de OEP de maneira a obrigar a que as alterações de velocidade sejam combinações lineares dos vectores de posições e velocidades, o algoritmo garante que as partículas iniciadas no hiperplano $\mathbf{P}$ nele se mantêm até o algoritmo terminar. As equações do OLEP são então:

$$\mathbf{v}_i^{t+1} = \mathbf{v}_i^t + u(0,\phi_1) \oplus (\mathbf{p}_i^t - \mathbf{x}_i^t) + u(0,\phi_2) \oplus (\mathbf{p}_g^t - \mathbf{x}_i^t) \tag{4.1}$$

$$\mathbf{x}_i^{t+1} = \mathbf{x}_i^t + \mathbf{v}_i^t \tag{4.2}$$

A diferença em relação às equações padrão está no facto de, em vez dos vectores de números aleatórios $\mathbf{u}(0,\phi_1)$ e $\mathbf{u}(0,\phi_2)$, termos as variáveis aletórias $u(0,\phi_1)$ e $u(0,\phi_2)$. A satisfação das restrições de domínio das variáveis α é garantida limitando o movimento das partículas ao hipercubo definido por aquelas restrições.

Esta abordagem à utilização de OLEP na resolução do problema de programação quadrática de treino das MVS, sendo baseada nos métodos de decomposição, necessita de um método de selecção de subconjuntos óptimos das variáveis de optimização e respectivos exemplos de treino. É este o subconjunto das variáveis que vai ser optimizado em cada iteração geral do algoritmo, enquanto o valor das restantes se mantém fixo.

Aqui é utilizado um critério baseado no declive da função objectivo, estando portanto dependente do facto da função de núcleo ser positiva semi-definida, resultando numa função objectivo com apenas um óptimo. Caso a função de núcleo não cumpra aquela propriedade, e a função objectivo seja multimodal, esta abordagem poderá ficar presa em qualquer óptimo local.

A abordagem foi testada num conjunto de dados de reconhecimento de caracteres, consistindo num conjunto de treino com 60000 imagens de dígitos, o que permite observar não só a viabilidade da abordagem, mas também o seu desempenho num problema de elevada dimensionalidade. Os resultados permitem concluir que os algoritmos de OEP podem ser utilizados com sucesso neste problema em particular, obtendo um desempenho comparável, em termos de classificação, com aqueles obtidos por dois algoritmos tradicionais de treino de MVS (OSM, SVMLight). Note-se no entanto que, em termos de complexidade temporal, os resultados experimentais sugerem que a OLEP necessita de muito mais tempo para fazer a optimização (superior a 10 vezes, em média). Por outro lado a abordagem evolucionária revelou escalar melhor, em termos de complexidade computacional, do que as outras abordagens.

Consequentemente, esta primeira abordagem evolucionária ao treino de máquinas de vectores de suporte, se por um lado permite concluir da viabilidade da aplicação dos algoritmos de OEP para a optimização de MVS com funções de núcleo definidas semi-positivas e problemas com elevado número de exemplos, dificilmente pode ser considerada competitiva com os métodos clássicos, não trazendo vantagens significativas

- além de uma maior facilidade de implementação - que possam justificar o seu uso generalizado. Em trabalho subsequente [Paquet and Engelbrecht, 2003a, 2007] foram identificadas outras fraquezas desta abordagem, sendo a mais significativa a do algoritmo alterado, em determinadas circunstâncias, poder convergir prematuramente, com as partículas a explorarem apenas a linha que as liga ao melhor ponto encontrado até à data, em vez de todo o hiperplano de soluções possíveis.

Alterações adicionais introduzidas nestes trabalhos permitem atenuar este problema, permitindo às partículas explorar o hiperplano em torno do melhor ponto encontrado, de maneira a garantir a convergência para um mínimo local, embora os algoritmos resultantes não pareçam promissores em ambientes com múltiplos óptimos, tal como é salientado pelos resultados experimentais apresentados. Adicionalmente, estas alterações apenas foram testadas em problemas *benchmark* genéricos de optimização com restrições lineares, não se conhecendo quais as consequências da sua aplicação ao treino de MVS em particular.

Ainda na mesma linha de abordagem, mas mais recentemente, Li et al [2007] introduziram na OLEP um mecanismo alternativo para evitar a convergência prematura do algoritmo, nomeadamente um operador de mutação adaptativo, aplicando-o ao treino de MVS para dois problemas *benchmark* de baixa dimensionalidade. Os resultados experimentais são, no entanto, pouco conclusivos, devido ao facto de os problemas serem relativamente simples e de as outras limitações da abordagem não serem tratadas, especialmente o facto de não ser utilizável com funções de núcleo não positivas semi-definidas.

Trabalhos sobre mecanismos de tratamento das restrições lineares, como [Monson and Seppi, 2005], sugerem que a preservação da validade das restrições através da OLEP, limitando as alterações de velocidade a combinações lineares das posições das partículas no enxame, podem não ser as mais adequadas. Com efeito, esta limitação diminui a capacidade exploradora do enxame ao reduzir as dimensões efectivas que podem ser exploradas ao número de partículas linearmente independentes (pensando nas coordenadas das partículas como vectores). À medida que as partículas convergem também essa dimensionalidade efectiva diminui, com algumas partículas a explorarem simplesmente uma trajectória linear que passa por $\mathbf{g}$, à medida que as restantes param. Monson and Seppi [2005] apontam ainda que mesmo a utilização de estratégias de aumento da diversidade podem ser de pouco uso efectivo nesta situação, já que são as próprias equações alteradas que sobre-restringem o comportamento exploratório do enxame.

4.3.2 Treino de MVS por estratégias evolutivas

A abordagem mais interessante ao treino evolucionário de MVS surge em Mierswa [2006a] e no trabalho subsequente do mesmo autor. Este é o primeiro a reconhecer as possibilidades dos algoritmos evolucionários como mecanismo de generalização das MVS, nomeadamente por permitirem o uso de funções de núcleo não positivas semi-definidas e a optimização multi-objectivo de critérios concorrentes. Mierswa [2006a] implementou um algoritmo evolucionário baseado em estratégias evolutivas e comparou resultados utilizando três operadores de mutação diferentes, que procuram capitalizar as características específicas do problema de treino de MVS, nomeadamente o facto de apenas alguns dos α_i serem diferentes de 0 na solução final, e normalmente serem C, para melhorar o desempenho do algoritmo de treino. As três abordagens diferem no seguinte:

- Na primeira variante é utilizado um operador de mutação gaussiana típico, no qual uma variável aleatória com distribuição normal e desvio padrão $C/10$ é adicionada aos indivíduos que codificam os α_i como valores reais. Uma recombinação com 0.9 de probabilidade foi também utilizada.
- Já na segunda abordagem a mutação é alterada de maneira a assemelhar-se à de um algoritmo genético clássico, alterando cada gene entre os valores de 0 e C com probabilidade 1/4. Esta versão capitaliza o facto de os α_i finais terem valores 0 ou C e muito poucos terem valores intermédios.
- A terceira variante é um misto das duas anteriores, procurando associar a velocidade da segunda à precisão da primeira.

Também foi implementado um algoritmo de treino utilizando um algoritmo padrão de optimização por enxames de partículas. Note-se que de maneira a simplificar o processo de treino o autor considera o problema de optimização com $b = 0$. Desta forma a restrição linear desaparece e os algoritmos podem ser utilizados sem alterações, embora com o custo de reduzir num grau de liberdade as possibilidades de optimização [Burges, 1998], obrigando a que todos os hiperplanos solução considerados durante o treino tenham obrigatoriamente de passar pela origem do espaço dos parâmetros.

Em termos de resultados obtidos, esta abordagem distancia-se significativamente da anterior. As quatro variantes evolucionárias foram testadas contra duas implementações clássicas (MySVM, LibSVM) utilizando um conjunto de problemas padrão de classificação. O desempenho em termos de precisão da classificação é semelhante entre as abordagens evolucionárias e as tradicionais, com a excepção da abordagem baseada em OEP, a qual o autor considera não proporcionar um desempenho preditivo comparável ao dos restantes algoritmos.

No que diz respeito à complexidade temporal das diferentes abordagens, embora a variante gaussiana se mantenha um grau de magnitude acima das abordagens tradicionais, as restantes versões, incluindo a que usa algoritmos de enxame, já são mais competitivas com as abordagens clássicas, sendo mesmo assim entre duas a quatro vezes mais lentas. Apesar de mais lenta, a variante gaussiana é a que o autor considera obter melhor desempenho global, sendo a que é utilizada em trabalho posterior. Em trabalho subsequente o mesmo autor faz ainda duas contribuições adicionais importantes para a área do treino evolucionário de máquinas de vectores de suporte.

Treino de MVS com funções de núcleo não positivas semi-definidas

Mierswa [2006b]; Mierswa and Morik [2008] comparam o desempenho da melhor variante evolucionária com as já referidas MySVM, LibSVM e uma implementação de máquinas de vectores de relevância[1] utilizando uma função de núcleo não positiva semi-definida - núcleo de Epanechnikov - num conjunto de problemas padrão. As máquinas de vectores de relevância são uma abordagem da aprendizagem estatísticas que procura lidar com núcleos não positivos [Tipping, 2001]. O desempenho classificativo do algoritmo evolucionário foi, em quase todos os problemas testados, superior ao dos métodos clássicos. Também superou em geral as máquinas de vectores de relevância, sendo que, adicionalmente, estas são de utilização impraticável para problemas reais, pelo facto do seu tempo de execução, mesmo para problemas muito pequenos, ser extremamente elevado (na ordem das dezenas de dias).

Estes são os únicos resultados que conhecemos onde fica demonstrada a viabilidade da utilização de um algoritmo evolucionário, neste caso baseado em estratégias evolutivas, no treino de MVS com funções de núcleo não positivas semi-definidas. Note-se ainda o bom desempenho e robustez das abordagens clássicas nestes problemas de optimização, possivelmente com mais de um óptimo, sendo que conseguem encontrar sempre soluções, mesmo que de qualidade inferior às encontradas pelos métodos evolucionários.

Treino multi-objectivo de MVS

Em [Mierswa, 2007] é proposta uma abordagem evolucionária multi-objectivo ao treino de MVS, na qual se separa explicitamente o termo relativo ao erro de classificação do termo relativo à optimização da complexidade do modelo preditivo. Assim, os dois termos da função a optimizar no treino de MVS são separados, deixando de ser necessário o parâmetro C. Um algoritmo evolucionário (de novo a variante baseada em estratégias evolutivas) é utilizado para optimizar as duas funções resultantes, produzindo uma frente de Pareto contendo todas as soluções com erro de classificação e complexidade

[1] Do inglês *relevance vector machines*.

do modelo mínimos. Esta pode ser utilizada para eficazmente evitar o problema do sobre-ajustamento do modelo na aprendizagem com MVS, sem a necessidade de utilizar explicitamente o parâmetro C para controlar o equilíbrio entre precisão e qualidade do modelo.

Estas duas contribuições, aliadas ao facto das melhores variantes implementadas já serem mais competitivas em termos de rapidez de execução quando comparadas com os algoritmos tradicionais, sublinham o interesse desta linha de investigação e estabelecem claramente a utilidade da utilização de algoritmos evolucionários no treino de MVS. A abordagem possui, no entanto, algumas limitações, mais especificamente por a variante com melhor desempenho ser algo lenta e por apenas serem conhecidos resultados para um único núcleo não PSD e um número relativamente pequeno de problemas de teste.

Finalmente, é interessante o facto de a abordagem baseada em OEP não ser competitiva em termos de desempenho preditivo, já que, sendo a mais rápida a convergir, poderia constitui uma solução eficiente, caso esse desempenho pudesse ser melhorado. Esta situação pode estar relacionado com problemas de convergência prematura do algoritmo, havendo aqui claro espaço para melhorias ou diferentes abordagens utilizando OEP.

4.3.3 Optimização evolucionária do problema primal

Ambas as abordagens descritas nas secções anteriores, bem como os métodos de optimização clássicos, se centram na optimização do problema dual. Uma terceira linha de investigação, apresentada em [Stoean et al, 2007], e, parcialmente, em [Jun and Oh, 2006], segue uma abordagem alternativa ao escolher uma representação diferente sobre a qual os algoritmos evolucionários irão actuar, nomeadamente escolhendo o problema primal como alvo da optimização.

As abordagens anteriores, baseadas no problema dual, usam os multiplicadores de Lagrange α como variáveis de optimização. Essa escolha não é arbitrária, antes assentando em critérios objectivos relacionados com a flexibilidade e complexidade computacional de ambas as formulações do problema. Mais especificamente, temos quatro factores que tornam a optimização do primal pouco apelativa:

1. A formulação do primal obriga a que o produto interno da equação 2.9 seja recalculado para todas as instâncias a cada iteração, o que não acontece no dual, levando a um custo computacional muito mais elevado para o cálculo da função objectivo.

2. No problema dual é também possível calcular a matriz de núcleo $\mathbf{K}$, com $K_{ij} = k(x_i, x_j)$, *a priori*, de maneira a não ter de voltar a lidar com os exemplos de treino

durante o processo de optimização, facto que também é impossível no primal.

3. Finalmente, esta abordagem torna toda a metodologia baseada em MVS muito menos geral do que as abordagens baseadas no problema dual. Como a função de classificação pode ser escrita sob a forma $f(\mathbf{z}) = \text{sgn}\left(\sum_i \alpha_i y_i k(\mathbf{x_i}, \mathbf{z}) + b\right)$, todo algoritmo fica dependente apenas da função de núcleo aplicada directamente às instâncias, o que é um aspecto essencial do chamado "truque" do núcleo. Este torna desnecessário o mapeamento explícito das instâncias para o novo espaço. No caso do primal, o produto interno $\langle \mathbf{w}, \mathbf{x_i} \rangle$ em cada restrição necessita explicitamente da instância transformada, i.e., é necessário conhecer explicitamente e aplicar a função de mapeamento Φ a cada instância. Esta limitação, na prática, impossibilita a aprendizagem eficiente usando funções de núcleo no caso não linear.

Independentemente das limitações apontadas, esta abordagem, utilizando um algoritmo genético com operadores padrão para representações reais e acrescentado um factor de penalização à função objectivo como forma de lidar com as restrições, foi utilizada com sucesso para optimizar directamente os parâmetros $\mathbf{w}$ e b em vários problemas de classificação padrão, sendo que os valores das variáveis de folga ξ_i são directamente calculados. Os autores sublinham que a maior vantagem desta abordagem consiste na possibilidade de aceder e interagir directamente com os parâmetros de decisão, o que permite observar a relevância de cada atributo para o hiperplano decisor.

Um elemento de mérito adicional desta linha de investigação consiste na aplicação da metodologia proposta num problema real, no sentido de que não faz parte das bibliotecas habituais de problemas padrão, tendo, pelo contrário, implicado a colaboração de peritos da área de aplicação [Stoean et al, 2011]. O problema em questão consiste na previsão do nível de fibrose do fígado em casos de hepatite C e a abordagem foi ampliada de forma a englobar um segundo nível de representação binário, o qual permitia co-evoluir o conjunto de atributos relevantes simultaneamente com a superfície de decisão. Os resultados experimentais mais uma vez demonstram a competitividade do algoritmo evolucionário, quando comparado com um método tradicional de treino de MVS.

4.3.4 Conclusões

A pesquisa bibliográfica torna claro que o treino evolucionário de MVS é uma área ainda pouco explorada, havendo basicamente três linhas de investigação recentes e um pequeno número de investigadores que nelas trabalham. Os resultados obtidos ainda são preliminares, embora permitam desde já vislumbrar as contribuições que a

abordagem pode trazer à área, bem como as dificuldades a ela inerentes. Numa análise mais detalhada, há vários pontos que podemos salientar:

- O treino de máquinas de vectores de suporte utilizando algoritmos evolucionários é viável tanto no problema primal como no dual, embora as vantagens da optimização do primeiro pareçam claramente ofuscadas pela perca de flexibilidade e custos computacionais acrescidos.

- Os resultados em termos de desempenho preditivo são competitivos para os problemas de teste padrão utilizados nos diversos trabalhos experimentais. Já em termos de complexidade temporal, as abordagens clássicas apresentam uma vantagem significativa, como seria de esperar. No entanto, tal como é demonstrado pela segunda linha de investigação que discutimos, existe claramente espaço para melhoria.

- A abordagem evolucionária com melhor desempenho aparente é baseada em estratégias evolutivas e procura utilizar algumas características do problema de maneira a ajustar os operadores à especificidade daquele. Curiosamente, os algoritmos de optimização por enxames de partículas, que frequentemente apresentam desempenhos favoráveis quando comparados com outras abordagem evolucionárias em problemas de optimização, apresentam problemas nas duas abordagens em que são utilizados. Na primeira são dificuldades de convergência, enquanto na segunda não são competitivos em termos de precisão classificativa.

- As contribuições mais significativas da abordagem evolucionária estão estabelecidas em princípio - utilização de núcleos não positivos semi-definidos e possibilidade de optimização multicritério - mas ainda não foi demonstrada a sua utilidade em problemas concretos. Por exemplo, ainda não há resultados experimentais onde um núcleo não positivo semi-definido, treinado por um algoritmo evolucionário, apresente melhores resultados em termos de classificação do que os núcleos habituais.

4.4 Questões em Aberto e Oportunidades de Investigação

A utilização de máquinas de vectores de suporte num problema específico apresenta três momentos importantes de decisão ao utilizador, os quais terão implicações significativas em termos do desempenho do classificador final. O utilizador tem de escolher qual o subconjunto de características que deve fornecer ao processo de treino, qual o núcleo a utilizar e respectivos parâmetros e que algoritmo vai ser usado para treinar a MVS.

Mais ainda, estas decisões são dependentes entre si. Para determinado subconjunto de características um núcleo pode funcionar melhor que outro e para o mesmo núcleo diferentes parâmetros podem ser mais adequados a determinadas características. Em relação à escolha do núcleo e do método de treino, até recentemente a escolha estava confinada a núcleos positivos semi-definidos e a um conjunto de algoritmos de treino bem conhecidos, os quais dependiam daquela característica dos núcleos para garantir o treino óptimo do classificador. Com a proposta de mecanismos de treino evolucionários para as MVS abre-se a perspectiva de utilização de núcleos não positivos semi-definidos, os quais podem ser mais adequados a determinado tipo de problema.

Nas secções anteriores descrevemos as diversas abordagens encontradas na bibliografia da computação evolucionária para estes três aspectos das máquinas de vectores de suporte. Ficou claro que a selecção de características, em especial em conjunção com a optimização dos parâmetros do núcleo, é a área mais tratada e com abordagens frutuosas. Já a síntese evolucionária de núcleos e o treino de MVS são áreas ainda relativamente por explorar, com poucas abordagens e estas com margens de melhoria significativas. Dos trabalhos encontrados fica no entanto clara a viabilidade da abordagem evolucionária a estas questões, com os resultados experimentais positivos apresentados na literatura a justificar a sua utilização.

Nesta dissertação propomos-nos debruçar sobre esta área, procurando contribuir para o estudo dos aspectos identificados como mais carenciados em termos de investigação anterior. Mais especificamente, procuraremos desenvolver algoritmos de treino mais eficientes e estudar a sua utilização em conjunto com núcleos não PSD. Outro aspecto que pretendemos explorar é a sua integração, pela primeira vez, num algoritmo de optimização de MVS em que todos os aspectos discutidos anteriormente, nomeadamente um núcleo adequado e os diversos parâmetros, possam ser encontrados automaticamente com intervenção mínima do utilizador e com um desempenho pelo menos competitivo com as metodologias clássicas.

Decidimos, neste trabalho, não incluir a selecção de características por dois motivos. Por um lado é uma vertente em que já existe muita investigação realizada, não só no que diz respeito às MVS, mas mesmo como área de estudo independente do algoritmo. Por outro lado essa selecção, como já foi discutido, pode sempre ser realizada num passo anterior ao treino do algoritmo de classificação propriamente dito.

Finalmente, em termos de computação evolucionária, optámos pela utilização de algoritmos de optimização por enxames de partículas. Esta escolha prende-se não apenas com o nosso conhecimento anterior destes algoritmos, mas também com o sucesso com que foram anteriormente aplicados a alguns dos problemas aqui discutidos. Estamos conscientes que também houve aspectos, como o treino de MVS, em que abordagens anteriores utilizando inteligência de enxame não se relevaram competitivas com outros algoritmos evolucionários. No entanto acreditámos, conhecendo a potencialidade

desses métodos, que se iriam revelar úteis aos nossos objectivos.

Capítulo 5

Optimização Predador-Presa com Batedores

O treino de uma máquina de vectores de suporte (MVS) é um problema de optimização numérica que consiste na escolha de um vector de valores óptimos α^* que permitam maximizar o valor da função (5.1). Esses valores estão sujeitos a dois tipos de restrições diferentes. As restrições (5.3) são restrições de domínio que se limitam a garantir que cada α_i^* se encontra no intervalo $[0, C]$, em que C é o parâmetro da MVS que controla o equilíbrio entre o erro e capacidade de generalização da MVS, medida pela margem entre o hiperplano de classificação e os vectores de suporte. A restrição (5.2) obriga a que o ponto α^* se encontre no hiperplano definido pela restrição.

$$\text{maximizar } \sum_i \alpha_i - \frac{1}{2} \sum_i \sum_j \alpha_i \alpha_j y_i y_j k(\mathbf{x}_i, \mathbf{x}_j), \tag{5.1}$$

$$\text{sujeito a } \sum_i \alpha_i y_i = 0 \tag{5.2}$$

$$\text{e } \forall i : 0 \leqslant \alpha_i \leqslant C. \tag{5.3}$$

Uma simplificação proposta inicialmente por Burges [1998], usada em várias abordagens tradicionais e nas mais recentes abordagens evolucionárias [Mierswa, 2006a; Mierswa and Morik, 2008], considera $b = 0$ na formulação inicial do problema (2.9), fazendo assim desaparecer esta última restrição. Esta simplificação obriga a que o hiperplano separador do problema primal passe obrigatoriamente pela origem do referencial. Esta limitação corresponde à perda de um único grau de liberdade na escolha do hiperplano, o que não é muito significativo quando o problema tem elevada dimensionalidade. Em termos experimentais parece não ter resultados negativos relativamente à precisão final do classificador, permitindo, no entanto, uma considerável simplificação dos algoritmos

de optimização, especialmente os evolucionários.

A função a optimizar é, como já sabemos, uma função quadrática, cuja forma é controlada pela função de núcleo $k(\mathbf{x}_i, \mathbf{x}_j)$. Na formulação habitual das máquinas de vectores de suporte a função de núcleo é positiva definida, garantindo uma função objectivo côncava com um único máximo global. Para este caso os métodos de treino determinista habitualmente utilizados, baseados em programação quadrática, são extremamente eficientes. Embora métodos populacionais estocásticos, como os algoritmos evolucionários, não sejam competitivos em termos de rapidez de optimização, é, no entanto, mesmo assim, interessante estudar o seu desempenho neste tipo de problemas por duas ordens de razões. Em primeiro lugar, a optimização do problema com um núcleo positivo definido, permite estimar o desempenho dos algoritmos evolucionários no problema mais complexo, no qual o núcleo não é positivo semi-definido. No mínimo, a abordagem do problema básico, permite estabelecer se o algoritmo evolucionário utilizado tem potencialidade para a optimização do problema mais difícil, i.e., caso o algoritmo evolucionário em consideração não consiga treinar um classificador competitivo quando o problema de optimização é unimodal, menos capaz será de o fazer quando houver vários óptimos possíveis. A possibilidade dos algoritmos evolucionários terem dificuldades na optimização do problema base está relacionada com a segunda ordem de razões de interesse dessa mesma optimização. Do ponto de vista da computação evolucionária, mesmo o problema base pode constituir um desafio significativo, já que apresenta várias dificuldades tradicionalmente complexas para muitos destes algoritmos:

1. Elevada dimensionalidade - Os algoritmos evolucionários, como muitos algoritmos de optimização, sofrem da chamada "maldição da dimensionalidade", a qual significa basicamente que o seu desempenho diminui substancialmente com o aumento do número de dimensões do problema a optimizar. Se relembrarmos que cada um dos parâmetros α_i está associado a um exemplo de treino do problema de classificação original, e que este facilmente pode conter milhares de exemplos, estaremos a considerar problemas com milhares de parâmetros a optimizar. Esta elevada dimensionalidade terá obrigatoriamente consequências em termos do desempenho dos algoritmos de evolucionários.

2. Localização do óptimo - Uma especificidade deste problema de optimização consiste no facto de muitos dos α_i^*, valores óptimos para os parâmetros a optimizar, terem valor 0. Este facto resulta de apenas um pequeno número de exemplos ser vector de suporte, correspondendo a um α_i^* diferente de 0. Dos α_i^* não nulos, alguns terão o valor C, o máximo valor permitido. Apenas os restantes possuirão um valor intermédio. Esta característica resulta em que, para uma larga maioria das dimensões, a solução do problema de optimização se encontre na extremidade do espaço de procura. Também aqui os algoritmos evolucionários poderão

encontrar dificuldades, já que muitos dos operadores neles utilizados privilegiam a procura na vizinhança do hiper-cubo que contém a população, com menor probabilidade de procura fora deste e, consequentemente nas fronteiras do espaço de procura.

3. Não separabilidade - A função objectivo é não separável, já que os produtos entre os diversos α_i^* implicam que a direcção de optimização pode ser influenciada por vários parâmetros simultaneamente. Funções não separáveis são normalmente mais difíceis de optimizar por algoritmos evolucionários do que as funções aditivamente separáveis.

Embora o treino de máquinas de vectores de suporte com núcleos PSD constitua só por si um desafio significativo para os algoritmos evolucionários, não deixa, no entanto, de ser apenas uma etapa para o nosso objectivo principal, que consiste no treino de MVS com núcleos não positivos definidos. Estes núcleos trazem um nova dificuldade ao problema de optimização, já que podem produzir funções objectivo com múltiplos óptimos locais. A forma da função e a localização dos óptimos são controlados pelo núcleo utilizado e pelos exemplos constituintes do conjunto de treino. Temos assim inúmeras funções possíveis, com características diferentes, para optimizar. Nestes problemas, os métodos tradicionais baseados em programação quadrática podem ser ultrapassados em termos de desempenho pelos algoritmos evolucionários, já que aqueles não se encontram preparados para lidar com múltiplos óptimos locais. Já os algoritmos evolucionários, sendo algoritmos de optimização global, têm tido bastante sucesso em termos de optimização multimodal. Claro que a existência de múltiplos óptimos locais vem acrescentar uma dificuldade adicional à lista anteriormente apontada para este problema de optimização.

A aplicação de algoritmos evolucionários a um novo problema implica, normalmente, a selecção de um algoritmo apropriado, a adequação ao problema em causa (por exemplo, através da definição de novos operadores) e a afinação dos seus parâmetros. Este processo pode ser demorado mesmo para um único problema simples. Quando se pretende avaliar um algoritmo evolucionário para toda uma classe de problemas, a abordagem tradicional implica a avaliação do algoritmo num subconjunto representativo dos problemas. No caso do treino de máquinas de vectores de suporte, onde a avaliação é feita através do cálculo do desempenho do classificador num conjunto de teste que pode conter centenas ou milhares de instâncias, este processo torna-se bastante demorado. Se acrescentarmos que uma avaliação razoavelmente precisa implica a utilização de avaliação cruzada *n-fold*, tipicamente com $n = 20$ ou mesmo 10 repetições de uma validação cruzada 10-*fold*, podemos estar a falar de 20 a 100 execuções do algoritmo de treino, por indivíduo, vezes o número de gerações, vezes o número de problemas a ser considerado. Este nível de complexidade computacional torna extremamente demorado o desenvolvimento de algoritmos evolucionários para o treino de MVS, factor provavel-

mente decisivo no facto de haver poucas abordagens evolucionárias na literatura para este problema específico.

De maneira a evitar parcialmente estas dificuldades, propomos, neste trabalho, seguir uma metodologia diferente, procedendo a uma primeira fase de teste e adequação dos algoritmos que pretendemos usar, utilizando não o problema objectivo mas um conjunto de funções genéricas de teste retiradas da bibliografia da optimização. Estas funções serão seleccionadas e, quando necessário, modificadas, de maneira a apresentar aos algoritmos de optimização as dificuldades que esperamos encontrar nos problemas reais de treino de máquinas de vectores de suporte. Para implementar esta metodologia iremos:

1. seleccionar algoritmos de optimização da nossa área alvo que tenham demonstrado ser promissores em pelo menos algumas das dificuldades que antevemos encontrar;
2. propor um novo algoritmo desenvolvido com o objectivo de responder a essas mesmas dificuldades;
3. definir um conjunto de problemas de teste que possam agir como problemas substitutos do nosso problema original;
4. comparar os diversos algoritmos nos diferentes cenários antevistos;
5. proceder às alterações que nos pareçam adequadas para melhor enfrentar o problema final.

Consideramos que esta metodologia apresenta várias vantagens quando comparada com as abordagens anteriores encontradas na bibliografia, onde a proposta e teste de novos algoritmos é feita directamente sobre problemas de classificação utilizando máquinas de vectores de suporte:

- A vantagem mais significativa será em termos computacionais, no sentido em que será possível testar mais variantes de algoritmos e mecanismos específicos para as dificuldades previstas no problema original. A avaliação de funções teste típicas de optimização numérica é computacionalmente muito mais barata do que a avaliação de uma MVS. No caso extremo dos problemas de muito alta dimensionalidade a experimentação é particularmente proibitiva.
- O treino de máquinas de vectores de suporte necessita de um conjunto de funcionalidades auxiliares que são demoradas de implementar, de maneira que é geralmente feito no contexto de pacotes de software alargados, tipicamente de *data mining* e frequentemente escritos em *Java*. A utilização destes problemas

substitutos implica que podemos escrever software específico para optimização numérica, significativamente mais simples, e que podemos seleccionar uma linguagem que resulte em executáveis rápidos (neste caso $C++$), com ganhos adicionais em termos de eficiência computacional.

- A experimentação alargada, que nos é permitida pelos ganhos de eficiência referidos anteriormente, permitirá um acrescido conhecimentos das dificuldades que se nos depararão no treino de MVS e, esperamos, dos mecanismos necessários para as ultrapassar.

- Várias das dificuldades que identificámos para o problema do treino de máquinas de vectores de suporte são comuns a muitos outros problemas de optimização. Os resultados obtidos neste contexto generalizado podem assim constituir contributos valiosos de investigação para a área mais genérica da optimização utilizando algoritmos evolucionários.

Obviamente que estes problemas não modelarão perfeitamente o problema original, de maneira que uma fase secundária de trabalho sobre problemas de classificação será necessária. No entanto chegaremos a essa fase com algoritmos de características já parcialmente adequadas ao problema, um melhor conhecimento dos obstáculos a enfrentar e ferramentas ou mecanismos que nos permitam evitá-los.

5.1 Os Algoritmos

Neste trabalho propomos-nos utilizar algoritmos de optimização por enxames de partículas (OEP) no treino de máquinas de vectores de suporte. Estes algoritmos basearam-se inicialmente numa metáfora inspirada pela observação das interacções sociais entre indivíduos em bandos de aves [Kennedy and Eberhart, 1995]. Verificou-se que o comportamento altamente coordenado de milhares de indivíduos era gerado por regras bastante simples. Mais especificamente, consegue-se reproduzir esse comportamento ao fazer cada ave seguir simultaneamente o seu próprio objectivo e um (ou vários) dos seus vizinhos.

Kennedy and Eberhart [1995] transformaram esta ideia num algoritmo de optimização, criando enxames em que cada partícula (ave) ocupa um ponto em $\mathbb{R}^m$, correspondendo a uma possível solução para um problema de optimização nesse mesmo espaço. O algoritmo usa como heurística principal a noção de que, se uma partícula se mover entre a melhor posição que encontrou e a melhor posição encontrada pelo enxame, irá explorar obrigatoriamente zonas promissoras do espaço de procura. Assim, novas melhores posições individuais serão encontradas, atraindo partículas vizinhas, e levando

o enxame a deslocar-se para regiões cada vez mais promissoras até que um óptimo da função a optimizar seja encontrado.

O resultado final é um algoritmo de optimização global estocástico, baseado numa população de soluções. Nesta população, chamada enxame, os indivíduos, chamados partículas, são representados por vectores de números reais no espaço multidimensional correspondente ao domínio da função a optimizar, também chamada função objectivo. A qualidade do indivíduo é medida pelo valor devolvido pela função objectivo para o ponto em causa.

Além da posição no espaço de procura, uma partícula possui também uma velocidade. Esta está dependente da intensidade com que a partícula é atraída pela sua melhor posição e a melhor posição dos vizinhos. O algoritmo de optimização por enxames de partículas procura uma solução para um problema específico, fazendo iterativamente alterações à velocidade de cada partícula, e, consequentemente, à sua posição. Desta forma é definida uma trajectória para cada partícula - e para o enxame - no espaço de procura.

Desde a sua introdução que o algoritmo de OEP tem sido aplicado com sucesso a inúmeros problemas em diferentes áreas. Estas incluem o projecto de antenas, computação gráfica e problemas de visualização, aplicações biomédicas, o projecto de redes eléctricas e muitas, muitas outras [Poli et al, 2008]. Entre as qualidades que levaram a esta popularidade podemos apontar:

- a simplicidade conceptual;
- a facilidade de implementação;
- o baixo custo computacional;
- a necessidade de populações pequenas;
- a rapidez de convergência;
- a facilidade de adaptação a novos domínios de aplicação;
- a possibilidade de hibridização com outras abordagens.

Embora possa haver outros algoritmos de optimização na área da computação evolucionária que reclamem melhor desempenho em termos de optimização, a combinação de simplicidade, flexibilidade e desempenho dos algoritmos de optimização por enxames de partículas garante que estes permanecem como primeira escolha em muitos contextos de utilização prática. Mesmo em termos de desempenho, existem classes de problemas em que os algoritmos de enxame superam claramente outras abordagens evolucionárias

bem conhecidas, como é claramente demonstrado pelo trabalho de Langdon and Poli [2007], onde é utilizada programação genética para evoluir problemas que apresentam dificuldades a vários algoritmos em comparação. De forma genérica, são estas mesmas qualidades que nos levaram a seleccionar os algoritmos de OEP como ferramenta ideal para a optimização de máquinas de vectores de suporte.

Apesar do sucesso e popularidade, o algoritmo básico de OEP possui também algumas falhas que se encontram já identificadas desde a sua introdução [Angeline, 1998; Shi and Eberhart, 1999]. As mais significativas são:

1. Dificuldade em controlar o equilíbrio entre exploração local e global - Normalmente é utilizado um coeficiente ligado à velocidade, o qual pode diminuir linearmente ao longo da execução do algoritmo, de maneira a garantir que a velocidade e consequentemente as oscilações das partículas diminuem, permitindo passar de uma fase de exploração global para uma fase de exploração local e posterior convergência. Uma formulação equivalente usa um factor de contrição para garantir o mesmo objectivo [Clerc, 2006].

2. Incapacidade de manter a diversidade no enxame - Após a convergência ocorrer, e caso tenha ocorrido em torno de um óptimo local, não existe qualquer mecanismo equivalente à mutação, presente em outros algoritmos evolucionários, o qual permita a reintrodução de alguma diversidade no enxame.

3. Degradação do desempenho em funções não-separáveis - Quando o nível de interacção entre as diversas variáveis é muito elevado, este algoritmo pode ter um desempenho inferior ao de outros algoritmos evolucionários, e.g., estratégias evolutivas.

4. Dificuldade em afinar boas soluções - Por vezes este tipo de algoritmo apresenta algumas dificuldades nos últimos estágios da optimização, i.e., quando uma boa solução muito próxima do óptimo global, mas que não é exactamente esse óptimo, foi encontrada e o algoritmo estagna.

Existe uma bibliografia extremamente rica de alterações ao algoritmo base, com as mais importantes resumidas em *overviews* da área [Banks et al, 2007, 2008; Poli et al, 2007]. Algumas das variantes contemporâneas mais bem sucedidas baseiam-se em abordagens híbridas, as quais usam um ou mais operadores de mutação em conjugação com o algoritmo de OEP para minorar as dificuldades atrás descritas [Gao and Xu, 2011a,b]. Abordagens meméticas, que combinam estratégias de procura local com as capacidades de exploração global do algoritmo de OEP foram também apresentadas, obtendo algum sucesso [Petalas et al, 2007].

A generalidade daquelas abordagens utiliza enxames homogéneos, i.e. enxames em que todas as partículas se comportam da mesma maneira, obedecendo ao mesmo conjunto de regras de actualização. No entanto, recentemente, tem aumentado o interesse em optimizadores por enxames de partículas heterogéneos, nos quais partículas dentro de um mesmo enxame podem apresentar comportamentos ou propriedades distintos [Engelbrecht, 2010; Montes de Oca et al, 2009]. Estas abordagens incluem o uso de diferentes topologias de vizinhança ou regras de actualização. Esta heterogeneidade permite que diferentes subconjuntos do enxame sejam afinados para diferentes aspectos do problema que se pretende optimizar ou para diferentes fases do processo de exploração.

Neste trabalho, com o objectivo de apresentar uma variante do algoritmo de optimização por enxames de partículas, simultaneamente capaz de responder às limitações listadas atrás e de se adequar facilmente ao problema que nos propusemos enfrentar, iremos descrever e analisar experimentalmente um novo algoritmo heterogéneo de optimização por enxames de partículas, denominado optimizador predador-presa com batedores (OPPB). Além das partículas habituais do enxame, o OPPB utiliza uma partícula extra, denominada predador, enquanto um subconjunto do enxame - os denominados batedores - são actualizados utilizando regras distintas.

Este novo algoritmo será testado utilizando um conjunto extenso de funções de teste e comparado com duas variantes de optimizadores por enxames de partículas e duas outras variantes de algoritmos baseados em evolução diferencial. Os algoritmos de OEP utilizados são:

- OEP clássico - com o objectivo de comparar o desempenho do novo algoritmo com uma versão padrão dos optimizadores por enxames de partículas, implementámos a versão do algoritmo clássico proposta por Shi and Eberhart [1999], com os parâmetros escolhidos de maneira a torná-lo equivalente à versão de convergência garantida proposta por Clerc [2006].

- OEP híbrido com procura moderadamente aleatória[1] (OEPHPMA) - Este é uma variante contemporânea do OEP introduzia por Gao and Xu [2011a], a qual apresenta resultados extremamente competitivos em muitas das funções de teste típicas da área, permitindo assim a comparação do nosso algoritmo com uma versão estado da arte do OEP.

Além das variantes de OEP, implementámos ainda duas versões de algoritmos baseados em evolução diferencial. Este algoritmo compartilha muitas das características dos OEP, incluindo a simplicidade conceptual e de implementação na sua forma básica,

[1] Do inglês *hybrid moderate random search particle swarm optimizer*.

bem como uma elevada popularidade na área da optimização por algoritmos evolucionários. A utilização destes algoritmos no nosso estudo experimental permite situar o desempenho do OPPB no contexto mais alargado dos algoritmos evolucionários contemporâneos. Os dois algoritmos baseados em ED utilizados são:

- ED clássica - também aqui, para efeitos de comparação, implementámos uma versão padrão do algoritmo [Storn and Price, 1997], mais especificamente a versão DE/$rand$/1/bin.
- ED com procura livre[2] (EDPL) - Esta é uma versão estado da arte do algoritmo, combinando elementos da evolução diferencial com a procura livre[3] [Penev and Littlefair, 2005] e a aprendizagem baseada em oposição[4] (ABO) [Tizhoosh, 2005] com resultados experimentais bastante promissores.

Convém aqui referir as razões porque não é incluída nesta comparação nenhuma instância de algoritmo evolucionário baseado em estratégias evolutivas, as quais fornecem alguns dos algoritmos de base evolucionária mais eficientes em optimização com uso disseminado, como por exemplo as estratégias evolutivas com adaptação da matriz de covariância [Beyer and Schwefel, 2002]. Além de constituírem um grupo de algoritmos com características bastante diferentes dos aqui apresentados, com forças e fraquezas distintas, existem duas razões pelas quais não utilizámos nenhuma estratégia evolutiva nesta comparação:

1. Já existe uma abordagem que realiza a optimização de máquinas de vectores de suporte utilizando estratégias evolutivas, com a qual pretendemos posteriormente comparar os algoritmos aqui desenvolvidos, no contexto já mais específico dos problemas de optimização.
2. O melhor representante desta categoria de algoritmos, a EE-AMC, possui uma complexidade espacial quadrática com o número de dimensões a optimizar, devido à necessidade de armazenar a matriz de covariância, o que torna a sua aplicação pouco prática para problemas com centenas ou milhares de variáveis, os quais facilmente ocorrem na optimização de máquinas de vectores de suporte.

Estes algoritmos são descritos detalhadamente nas próximas secções. Posteriormente serão comparados experimentalmente utilizando um conjunto de 16 funções de teste cuidadosamente seleccionadas para ilustrar as suas características específicas e que apresentaremos na última secção deste capítulo. Os resultados dessa comparação e a sua discussão serão finalmente apresentados no capítulo seguinte.

[2]Do inglês *free search differential evolution*.
[3]Do inglês *free search*.
[4]Do inglês *opposition based learning*.

5.2 Algoritmo Básico de Optimização por Enxames de Partículas

Na optimização por enxames de partículas cada elemento do enxame é representado por três vectores reais de tamanho m, assumindo um problema de optimização $f(\mathbf{x})$ em $\mathbb{R}^m$. Para cada partícula i temos um vector $\mathbf{x}_i$ que armazena a posição actual do indivíduo no espaço de procura. A posição correspondente ao melhor valor de $f(\mathbf{x})$ encontrado pela partícula até ao momento é guardado em $\mathbf{p}_i$. Finalmente, um terceiro vector $\mathbf{v}_i$ é utilizado para manter a velocidade da partícula.

O algoritmo começa com uma fase de inicialização (linhas 1-7 do algoritmo 5.1) na qual os vectores $\mathbf{x}_i$ são criados aleatoriamente, com distribuição uniforme, dentro do domínio da função a optimizar, armazenando-se em g o índice da partícula com menor avaliação (valor de $f(\mathbf{x})$), considerando um problema de minimização. A velocidade $\mathbf{v}_i$ é também inicializada aleatoriamente, com distribuição uniforme, mas no intervalo entre a amplitude negativa e positiva do espaço de procura, i.e., $\mathbf{u}(-|x_{max} - x_{min}|, |x_{max} - x_{min}|)$.

O ciclo principal deste algoritmo (linhas 8 a 22 do algoritmo 5.1) consiste essencialmente na repetição das regras de actualização de velocidade e posição para cada partícula até uma das condições de paragem ser atingida. Nas nossas implementações, os algoritmos terminam quando um determinado número de iterações t_{max} ou de avaliações c_{max} da função objectivo são atingidas, embora outras condições, como por exemplo uma determinada distância ao óptimo global ou um intervalo máximo de estagnação do algoritmo, possam também ser utilizadas.

Esta implementação utiliza as equações originais (5.4), com os parâmetros recomendados por Shi and Eberhart [1999], nomeadamente em termos dos valores de w, isto é utilizamos um peso associado à velocidade anterior, o qual decresce linearmente ao longo da execução do algoritmo. Nestas equações $(\mathbf{p}_i - \mathbf{x}_i^t)$ representa a distância entre a posição actual de uma partícula e a melhor posição que encontrou em iterações anteriores; $(\mathbf{p}_g - \mathbf{x}_i^t)$ representa a distância entre a posição actual da partícula e a melhor posição encontrada pelas partículas da sua vizinhança, $\mathbf{p}_g$; $\mathbf{u}(0, \phi_1)$ e $\mathbf{u}(0, \phi_2)$ são vectores de números aleatórios com distribuições uniformes entre 0 e ϕ_1 e 0 e ϕ_2, respectivamente; $\otimes$ denota a multiplicação dos vectores componente a componente.

Os parâmetros ϕ_1 e ϕ_2 permitem controlar a contribuição das componentes individual e social para a actualização da velocidade das partículas. Geralmente têm valores iguais, tipicamente com $\phi_1 = \phi_2 = 2$, garantindo que uma partícula é atraída com a mesma intensidade para o seu melhor e para o melhor da população, mantendo assim um movimento oscilante entre essas posições. Embora esta escolha de parâmetros seja muito comum e obtenha bons resultados para uma grande variedade de problemas, não

Algoritmo 5.1 Algoritmo básico de optimização por enxames de partículas.

para cada partícula i **faz**
  $\mathbf{p}_i \leftarrow \mathbf{x}_i \leftarrow \mathbf{u}(x_{min}, x_{max})$
  se $f(\mathbf{x}_i) < f(\mathbf{x}_g)$ **então**
    $g \leftarrow i$
  fim se
  $\mathbf{v}_i \leftarrow \mathbf{u}(-|x_{max} - x_{min}|, |x_{max} - x_{min}|)$
fim para
enquanto $t < t_{max}$ **e** $c < c_{max}$ **faz**
  para cada partícula i **faz**
    para cada dimensão d **faz**
      $v_{id} \leftarrow \omega v_{id} + u(0, \phi_1)(p_{id} - x_{id}) + u(0, \phi_2)(p_{gd} - x_{id})$
      $x_{id} \leftarrow x_{id} + v_{id}$
      $x_{id} \leftarrow \text{limita}(x_{id}, x_{min}, x_{max})$
    fim para
    se $f(\mathbf{x}_i) < f(\mathbf{p}_i)$ **então**
      $\mathbf{p_i} \leftarrow \mathbf{x}_i$
    fim se
    se $f(\mathbf{x}_i) < f(\mathbf{p}_g)$ **então**
      $g \leftarrow i$
    fim se
  fim para
fim enquanto

é obviamente universal, havendo investigação substancial sobre a escolha dos melhores parâmetros possíveis para cada problema [Pedersen, 2010b]. Neste trabalho, tanto para este como para os restantes algoritmos, optámos sempre pelo critério de utilizar os parâmetros padrão recomendados pelos autores para cada algoritmo, de maneira a minimizar o efeito da escolha dos parâmetros no desempenho dos algoritmos.

$$\begin{aligned} \mathbf{v}_i^{t+1} &= w\mathbf{v}_i^t + \mathbf{u}(0, \phi_1) \otimes (\mathbf{p}_i - \mathbf{x}_i^t) + \mathbf{u}(0, \phi_2) \otimes (\mathbf{p}_g - \mathbf{x}_i^t) \\ \mathbf{x}_i^{t+1} &= \mathbf{x}_i^t + \mathbf{v}_i^{t+1} \end{aligned} \tag{5.4}$$

Note-se que embora a abordagem aqui utilizada não utilize as equações canónicas (5.5) propostas por Clerc [2006], o desempenho dos algoritmos é semelhante, já que, com uma escolha adequada dos parâmetros, as formulações são equivalentes. Adicionalmente, embora a utilização do factor de constrição χ garanta a convergência do algoritmo, na prática esta também pode ser promovida pela utilização do peso w e do seu decremento linear. A escolha de umas regras de optimização em detrimento das outras teve apenas

a ver com a nossa experiência anterior de utilização destes algoritmos, segundo a qual o desempenho da versão com peso linear é geralmente superior ao da versão constrita.

$$\begin{aligned} \mathbf{v}_i^{t+1} &= \chi(\mathbf{v}_i^t + \mathbf{u}(0,\phi_1) \oplus (\mathbf{p}_i^t - \mathbf{x}_i^t) + \mathbf{u}(0,\phi_2) \oplus (\mathbf{p}_g^t - \mathbf{x}_i^t)) \\ \mathbf{x}_i^{t+1} &= \mathbf{x}_i^t + \mathbf{v}_i^{t+1} \\ \text{com } \phi &= \phi_1 + \phi_2 > 4, \\ \text{e } \chi &= \frac{2}{\phi - 2 + \sqrt{\phi^2 - 4\phi}} \end{aligned} \tag{5.5}$$

Dentro do ciclo de actualização (linhas 9 a 21 do algoritmo 5.1), inicialmente é calculada a velocidade da partícula $\mathbf{v}_i$, a qual é de seguida utilizada para actualizar a respectiva posição $\mathbf{x}_i$. A função *limita* é utilizada para manter os valores de $\mathbf{x}_i$ dentro dos limites permitidos para o problema. Após o cálculo da nova posição da partícula, é calculado o valor de $f(\mathbf{x}_i)$ e actualizados os valores de $\mathbf{p}_i$ e g, caso, respectivamente, a nova posição seja a melhor encontrada pela partícula ou a melhor solução encontrada por todas as partículas na vizinhança desta. Note-se que, nos resultados apresentados, utilizamos sempre uma vizinhança global, i.e., a vizinhança de cada partícula é constituída por todas as outras partículas do enxame.

5.3 OEP Híbrido com Procura Moderadamente Aleatória

O algoritmo estado da arte baseado em optimização por enxames de partículas que escolhemos para comparar com a nossa abordagem denomina-se OEP Híbrido com Procura Moderadamente Aleatória. Como já mencionámos, a escolha deste algoritmo teve sobretudo em conta o facto de, experimentalmente, com um conjunto bastante alargado de funções de teste e em comparação com outras versões de OEP e algoritmos evolucionários, este algoritmo ter apresentado resultados muito competitivos [Gao and Xu, 2011a]. Adicionalmente, trata-se de uma abordagem bastante recente e para a qual foi possível, com a nossa própria implementação, reproduzir os resultados apresentados na publicação original.

O OEPHMPA introduz alterações importantes no OEP original, tanto ao nível das equações de actualização, como em termos de hibridização, utilizando operadores de mutação adicionais. O algoritmo começa com uma fase de inicialização em tudo semelhante à do OEP básico (linhas 1-6 do algoritmo 5.2). Já a fase de actualização das partículas é substancialmente diferente. A nova posição de uma partícula começa por ser calculada utilizando a equação (5.6), a qual define um ponto de atracção $\mathbf{x}'_i$

dependente apenas da melhor posição encontrada pela partícula, $\mathbf{p}_i$, e da melhor posição da vizinhança, $\mathbf{p}_g$, i.e. a posição actual não é tida em conta. Este ponto de atracção é escolhido por, segundo os autores, a sua vizinhança coincidir com a zona mais promissora do espaço de procura.

$$\begin{aligned} x'_d &= r * p_{id} + (1 - r) * p_{gd}, \\ \text{com } r &= u(0, 1). \end{aligned} \tag{5.6}$$

Esta nova posição $\mathbf{x}'$ é posteriormente perturbada utilizando a equação (5.7). Ao ponto de atracção calculado anteriormente é adicionado um termo dependente da diferença entre a posição actual da partícula e o ponto médio do enxame, calculado usando a equação (5.9). A intensidade desta perturbação depende do valor do parâmetro fixo α e de um valor aleatório γ, distribuído segundo a equação (5.8). Esta distribuição foi definida de forma a garantir uma procura aleatória geralmente em torno de $\mathbf{x}'_i$, já que cerca de 80% dos valores de γ estão no intervalo $[-1, 1]$, embora valores muito mais altos possam surgir. Estes últimos são particularmente úteis em termos de manutenção da diversidade do enxame, garantindo que as partículas podem surgir em qualquer ponto do espaço de procura, o que, segundo os autores, melhora as suas capacidades em termos de procura global. O parâmetro α permite controlar a amplitude da perturbação, podendo ser utilizado para controlar a velocidade de convergência do algoritmo, à imagem do parâmetro w no OEP básico.

$$x_{id}^{t+1} = x'_d + \alpha\gamma(m_d - x_{id}^t), \tag{5.7}$$

$$\text{com } \gamma = \frac{u(0, 1)u(0, 1)}{u(-1, 1)} \tag{5.8}$$

$$\text{e } m_d = \frac{\sum_i^n p_{id}}{n}. \tag{5.9}$$

Os autores chamam a esta formulação do algoritmo de optimização por enxames de partículas, resumida no algoritmo 5.2, OEP com procura moderadamente aleatória. Esta permite ganhos substanciais quando comparada com o algoritmo básico, especialmente em termos de velocidade de convergência, mas ainda sofre de alguns dos problemas típicos da OEP, nomeadamente por não possuir mecanismos que permitam a fuga a uma situação de convergência prematura. Com efeito, é fácil de ver que quando o enxame converge e, consequentemente, $\mathbf{p}_i$ está muito próximo de $\mathbf{p}_g$, e os valores de $(m_d - x_{id})$ são 0, já que todas as partículas ocupam aproximadamente a mesma posição, nenhum mecanismo nas regras de actualização permite às partículas escapar ao ponto de atracção. Caso o enxame tenha prematuramente convergido para um óptimo local, já não será possível o recomeço do processo de exploração, excepto com a reinicialização do

Algoritmo 5.2 OEPHPMA - Movimento das partículas.

1: **para cada** partícula i **faz**
2: $\mathbf{p}_i \leftarrow \mathbf{x}_i \leftarrow \mathbf{u}(x_{min}, x_{max})$
3: **se** $f(\mathbf{x}_i) < f(\mathbf{x}_g)$ **então**
4: $g \leftarrow i$
5: **fim se**
6: $\mathbf{v}_i \leftarrow \mathbf{u}(-|x_{max} - x_{min}|, |x_{max} - x_{min}|)$
7: **fim para**
8: **enquanto** $t < t_{max}$ **e** $c < c_{max}$ **faz**
9: **para cada** partícula i **faz**
10: **para cada** dimensão d **faz**
11: $r \leftarrow u(0,1)$
12: $x'_d \leftarrow r * p_{id} + (1 - r) * p_{gd}$
13: $m_d \leftarrow \frac{\sum_i^n p_{id}}{n}$
14: $\gamma \leftarrow \frac{u(0,1)u(0,1)}{u(-1,1)}$
15: $x_{id}^{t+1} \leftarrow x'_d + \alpha\gamma(m_d - x_{id}t)$
16: $x_{id}^{t+1} \leftarrow \text{limita}(x_{id}^{t+1}, x_{min}, x_{max})$
17: **fim para**
18: **se** $f(\mathbf{x}_i^{t+1}) < f(\mathbf{p}_i)$ **então**
19: $\mathbf{p}_i \leftarrow \mathbf{x}_i^{t+1}$
20: **fim se**
21: **se** $f(\mathbf{x}_i^{t+1}) < f(\mathbf{p}_g)$ **então**
22: $g \leftarrow i$
23: **fim se**
24: **fim para**
25: **fim enquanto**

algoritmo. Os autores recomendam a diminuição linear deste parâmetro, entre 4.5 e 3.5, durante a execução do algoritmo.

Para ultrapassar este problema foram adicionados ao algoritmo dois operadores de mutação, cuja aplicação é descrita no algoritmo 5.3. A aplicação dos operadores de mutação em cada dimensão d é controlada por um parâmetro de limiar t_d e um parâmetro adicional m. Quando numa partícula i se verifica que $|x_{id}^t - x_{id}^{t+1}| < t_d$, a mutação deve ser aplicada na dimensão d.

O valor de t_d é extremamente importante, já que controla a frequência de aplicação dos operadores de mutação. Como tal, é utilizada uma calendarização que permite ir diminuindo o valor de t_d ao longo da execução do algoritmo. A regra de actualização de t_d é apresentada nas linhas 28-30 do algoritmo 5.3. Basicamente, o valor de t_d passa a t_d/m quando um contador de mutações realizadas na dimensão d, f_d, atinge um limite

Algoritmo 5.3 OEPHPMA - Mutações.

```
...
...
...
para cada partícula i faz
  para cada dimensão d faz
    se |x_id^t − x_id^{t+1}| < t_d então
      se f_d ≤ k/2 então
        x_id^{t+1} ← u(−1, 1)x_max
        se f(x_i^{t+1}) < f(p_i) então
          p_i ← x_i^{t+1}
        fim se
        se f(x_i^{t+1}) < f(p_g) então
          g ← i
        fim se
      senão
        p'_id ← p_id (1 + n(0, 1))
        p'_id ← limita(p'_id, x_min, x_max)
        se f(p'_i) < f(p_i) então
          p_i ← p'_i
        fim se
        se f(p'_i) < f(p_g) então
          g ← i
        fim se
      fim se
      f_d ← f_d + 1
    fim se
    se f_d = k então
      f_d ← 0
      t_d ← t_d/m
    fim se
  fim para
fim para
...
...
...
```

k. O valor inicial de t_d é x_{max} e os valores recomendados respectivamente para m e k são 10 e 40.

O primeiro dos operadores de mutação é um operador de mutação global, no qual o valor de x_{id} é substituído utilizando a equação (5.10). Trata-se de um operador de mutação global, na medida em que o valor da partícula na dimensão d pode assumir qualquer valor no seu domínio quando o operador é aplicado. Como a aplicação do operador pode ocorrer nos mesmos moldes durante toda a execução do algoritmo, dependendo apenas a sua frequência de aplicação de f_d, fica assim garantida a introdução de diversidade, mesmo após a convergência do algoritmo.

$$x_{id}^{t+1} = u(-1, 1)x_{max} \tag{5.10}$$

O segundo operador realiza uma procura local em torno da posição actual da partícula, adicionando perturbações à melhor posição encontrada pela partícula segundo a equação (5.11). Note-se a utilização de uma distribuição uniforme no primeiro operador, garantindo que os novos valores podem ocorrer com a mesma probabilidade em qualquer ponto do domínio, enquanto que neste segundo operador é utilizada uma distribuição normal, forçando assim a maioria dos valores a ocorrerem na vizinhança do valor de p_{id}. A utilização de dois operadores de mutação, com características mais comuns nos algoritmos evolucionários do que nos algoritmos de enxame, define a característica híbrida do algoritmo utilizada na sua denominação final: OEP híbrido com procura moderadamente aleatória.

$$p'_{id} = p_{id}\,(1 + n(0, 1)) \tag{5.11}$$

A utilização conjunta de todos os mecanismos descritos permite a este algoritmo apresentar resultados experimentais que são substancialmente melhores, não só do que os do algoritmo clássico, mas também do que os de várias variantes recentes deste e mesmo de outros algoritmos evolucionários, tal como é comprovado pelos resultados experimentais apresentados em [Gao and Xu, 2011a]. Estes resultados são inclusivamente os melhores que encontrámos entre variantes igualmente recentes apresentadas na literatura da área, especificamente utilizando um conjunto de funções de teste igualmente substancial (neste caso, 13 funções).

5.4 Algoritmo Predador-Presa

Nesta secção apresentamos a nossa proposta de OEP, o algoritmo de optimização predador-presa com batedores. Tal como o algoritmo anterior, procuramos responder a

algumas das limitações do OEP básico através da introdução de novos mecanismos. Ao contrário dessa abordagem, no entanto, procuramos manter-nos próximos da metáfora inicial da inteligência de enxame, evitando uma hibridização demasiado abrangente com outros algoritmos evolucionários, a qual dificulta a compreensão dos contributos que cada paradigma traz ao algoritmo final.

Este novo algoritmo acrescenta dois mecanismos ao algoritmo básico, sendo que ambos facilmente assentam em comportamentos biológicos comuns em comunidades de animais ou enxames de insectos. Ambos os mecanismos adicionam funcionalidades complementares ao algoritmo de OEP. O mecanismo predador-presa baseia-se nas interacções entre o enxame ou bando e os seus predadores e é utilizado para controlar a convergência prematura e manter a diversidade no enxame.

As partículas batedoras, ou batedores, reflectem comportamentos de exploração comuns a muitos enxames (formigas, abelhas). Nestes, indivíduos específicos realizam tarefas de exploração que não são partilhadas pelo resto do enxame, mas cujos resultados (uma nova fonte de alimento, por exemplo) são posteriormente aproveitados pelos restantes elementos do grupo.

No OPPB, as partículas batedoras adicionam um mecanismo simples de extensão dos comportamentos do enxame, cumprindo um objectivo duplo. Por um lado permitem introduzir no algoritmo informação especifica sobre um determinado problema, adaptando o algoritmo às características desse problema. Por outro lado, permitem também adicionar mecanismos de melhoria generalizada, sem alterar directamente o comportamento geral do enxame, já que apenas algumas partículas têm as suas regras de actualização modificadas.

Tanto as partículas predadoras como as partículas batedoras constituem formas de heterogeneidade neste algoritmo, no sentido de que não compartilham as mesmas regras de actualização das restantes partículas do enxame, logo o enxame não é homogéneo. A nossa preferência pela heterogeneidade em detrimento da alteração das equações base, como é feita nas técnicas de hibridização ilustradas pelo algoritmo da secção anterior, resulta da constatação de que é mais fácil adicionar novos comportamentos ao enxame, e estudar as interacções com os já existentes, através da introdução de novas partículas do que pela alteração de todas as existentes. A comunicação entre partículas é feita exclusivamente pelo acesso à melhor posição de outras partículas, tal como no algoritmo tradicional.

O algoritmo predador-presa, ainda sem as partículas batedoras, é apresentado no algoritmo 5.4. Pode-se aí verificar que a estrutura base é semelhante à do algoritmo de OEP clássico, incluindo a mesma fase de inicialização já descrita e a utilização das equações de actualização (5.4) para a generalidade das partículas.

Algoritmo 5.4 Algoritmo predador-presa.

para cada partícula i **faz**
  $\mathbf{p}_i \leftarrow \mathbf{x}_i \leftarrow \mathbf{u}(x_{min}, x_{max})$
  se $f(\mathbf{p}_i) < f(\mathbf{p}_g)$ **então**
    $g \leftarrow i$
  fim se
  $\mathbf{v}_i \leftarrow \mathbf{u}(-|x_{max} - x_{min}|, |x_{max} - x_{min}|)$
fim para
$\mathbf{x}_p \leftarrow \mathbf{u}(x_{min}, x_{max})$
$\mathbf{v}_p \leftarrow \mathbf{u}(-|x_{max} - x_{min}|, |x_{max} - x_{min}|)$
enquanto $t < t_{max}$ **e** $c < c_{max}$ **faz**
  para cada partícula i **faz**
    para cada dimensão d **faz**
      $v_{id} \leftarrow \omega v_{id} + u(0, \phi_1)(p_{id} - x_{id}) + u(0, \phi_2)(p_{gd} - x_{id})$
      se $u(0,1) < r \exp^{-|x_{id} - x_{pj}|}$ **então**
        $v_{id} \leftarrow v_{id} + u(-1, 1)|x_{max} - x_{min}|$
      fim se
      $v_{id} \leftarrow \text{limita}(v_{id}, -|x_{max} - x_{min}|, |x_{max} - x_{min}|)$
      $x_{id} \leftarrow x_{id} + v_{id}$
      $x_{id} \leftarrow \text{limita}(x_{id}, x_{min}, x_{max})$
    fim para
    se $f(\mathbf{x}_i) < f(\mathbf{p}_i)$ **então**
      $\mathbf{p}_i \leftarrow \mathbf{x}_i$
    fim se
    se $f(\mathbf{x}_i) < f(\mathbf{p}_g)$ **então**
      $g \leftarrow i$
    fim se
  fim para
  para cada dimensão d **faz**
    $v_{pd} \leftarrow \omega v_{pd} + u(0, \phi_1)(x_{gd} - x_{pd}) + u(0, \phi_2)(p_{gd} - x_{pd})$
    $x_{pd} \leftarrow x_{pd} + v_{pd}$
  fim para
fim enquanto

Uma ligeira diferença na aplicação destas equações consiste em que, no OPP, w é geralmente utilizado com valores substancialmente inferiores aos do algoritmo base, tipicamente variando linearmente entre 0.4 e 0.2. À imagem do parâmetro α no algoritmo anterior, um valor baixo para w obriga o enxame a convergir mais rapidamente, ficando este assim dependente dos mecanismos adicionais para evitar uma convergência prematura. Nas próximas duas subsecções descreveremos esses mecanismos detalhada-

mente.

5.4.1 O mecanismo predador-presa

Como já referimos anteriormente, uma das limitações do OEP clássico é a sua incapacidade para re-introduzir diversidade no enxame após este ter convergido para um óptimo local. À medida que os termos $(\mathbf{p}_i - \mathbf{x}_i^t)$ e $(\mathbf{p}_g - \mathbf{x}_i^t)$ se aproximam de 0, i.e., as partículas convergem, resta apenas o valor residual da velocidade, cada vez mais pequeno, para fazer mover as partículas. Desta forma, e como não existe nenhum mecanismo semelhante aos operadores de mutação presentes nos algoritmos evolucionários, depois de as partículas encontrarem uma zona promissora do espaço de procura e para ela convergirem, atingindo um óptimo, não existe forma da velocidade voltar a aumentar, permitindo ao enxame escapar desse óptimo.

Este é o comportamento desejável quando o óptimo encontrado é global, mas, no caso de ser apenas um óptimo local, não há maneira de o enxame voltar para uma fase de exploração global após entrar numa fase de exploração local. Adicionalmente, quando o enxame tem de subir um longo declive na função a optimizar, pode ocorrer um fenómeno de estagnação em que, apesar de longe do óptimo, as partículas já se encontram demasiado perto umas das outras para que as alterações à velocidade sejam suficientes para atingir o óptimo em tempo útil.

Introduzimos originalmente a ideia de um mecanismo predador-presa em [Silva et al, 2002b], com o objectivo de aliviar estes problemas, sendo que aqui apresentamos uma versão actualizada e simplificada do algoritmo resultante. A ideia é inspirada na imagem comum do predador em perseguição a um grupo de animais, produzindo dispersão no bando a um dado momento, sendo que este se reagrupa posteriormente, mantendo sempre alguma forma de coordenação. Instâncias biológicas típicas deste mecanismo podem ser vistas na perseguição de um leão a um grupo de gazelas ou de um atum a um cardume de sardinhas.

Como veremos posteriormente, esta inspiração biológica permitiu-nos criar um mecanismo que, mantendo a metáfora de inteligência de enxame do algoritmo, não só permite gerir a diversidade no enxame, como o permite fazer de forma naturalmente adaptativa, o que constitui uma vantagem sobre os algoritmos híbridos que empregam operadores de mutação tradicionais.

No nosso algoritmo, o predador é modelado como uma partícula extra, cuja velocidade $\mathbf{v}_p$ é actualizada através da equação (5.12), a qual é semelhante às equações de actualização das restantes partículas. A diferença mais substancial entre as equações reside no facto do predador ser atraído pela posição actual da melhor partícula, $\mathbf{x}_g^t$, em substituição da sua anterior melhor posição, i.e., a melhor posição anterior do predador não

entra na actualização da sua velocidade (nem sequer é mantida, já que o predador não é avaliado). Esta regra de actualização faz com que o predador oscile entre a posição actual e a melhor posição anterior da partícula com melhor desempenho do enxame, levando-o a efectivamente perseguir essa partícula no espaço de procura.

$$\mathbf{v}_p^{t+1} = w\mathbf{v}_p^t + \mathbf{u}(0,\phi_1) \otimes (\mathbf{x}_g^t - \mathbf{x}_p^t) + \mathbf{u}(0,\phi_2) \otimes (\mathbf{p}_g - \mathbf{x}_p^t) \tag{5.12}$$

O objectivo da partícula predador no algoritmo consiste em introduzir um factor de perturbação no enxame e garantir que este distúrbio aumenta à medida que o enxame converge para um único ponto. Para conseguir esse comportamento, adicionamos uma perturbação à velocidade das partículas numa dimensão j, conforme definido pela equação (5.13), onde $u(-1,1)$ e $u(0,1)$ são números aleatórios uniformemente distribuída entre os argumentos; x_{max} e x_{min} são, respectivamente, o limite superior e inferior para o espaço de procura e r é a probabilidade de perturbação definida pelo utilizador.

$$v_{ij}^t = v_{ij}^t + u(-1,1)|x_{max} - x_{min}|, \text{if } u(0,1) < r \exp^{-|x_{ij} - x_{pj}|} \tag{5.13}$$

Da equação (5.13) resulta que uma perturbação aleatória é adicionada ao valor da velocidade na dimensão j, com uma probabilidade que depende da distância da partícula ao predador nessa mesma dimensão. Esta probabilidade é máxima (r) quando a distância é 0, mas diminui rapidamente se a partícula escapa ao predador, i.e., se a distância entre eles aumenta. Uma vez que o predador persegue a melhor das partículas, a perturbação no enxame é mais provável quando todas as partículas estão muito próximas, ou seja, durante a fase de exploração local, e torna-se quase inexistente quando as partículas estão muito distantes. Este mecanismo permite às partículas escapar e encontrar um novo óptimo longe do atractor actual, mesmo nas últimas fases de exploração local.

Convém salientar de novo a capacidade adaptativa deste mecanismo, a qual permite que a intensidade do efeito do predador vá variando conforme este se encontre mais perto ou mais longe das restantes partículas, não havendo assim perturbações desnecessárias do enxame quando este está longe de convergir. Outro aspecto em que este mecanismo difere da utilização dos mais comuns operadores de mutação, é no facto de a perturbação ser realizada sobre a velocidade e não sobre a posição da partícula. O efeito é assim mais duradouro já que, através da componente da velocidade anterior nas equações de actualização das partículas, irá manter a partícula a oscilar durante algumas iterações.

O parâmetro r permite controlar a intensidade do efeito, sendo que nas experiências que realizámos esse valor foi mantido em 0.0008 - aproximadamente $1/(m \times 30)$. Note-se que, embora fora do âmbito deste trabalho, os parâmetros w, ϕ_1 e ϕ_2, que aqui se assume terem os mesmos valores que nas equações de actualização das restantes

partículas, podem assumir valores diferentes. Isto permite, por exemplo, tornar o predador mais lento, o que daria mais tempo ao enxame para convergir antes do efeito de perturbação se intensificar. O mecanismo predador-presa pode assim ser ajustado a problemas específicos, o que nos poderá vir a ser útil mais tarde.

5.4.2 Partículas batedoras

O optimizador predador-presa, só por si, é já um OEP heterogéneo, uma vez que a partícula predador é actualizada usando uma equação diferente da utilizada pelo enxame. No OPPB existe, no entanto, um segundo nível de heterogeneidade, dada a inclusão de partículas batedoras no enxame. As partículas batedoras, ou batedores, são um subconjunto do enxame que implementa estratégias de exploração diferentes da utilizada pelo enxame principal. Tal como o predador, têm uma inspiração biológica, já que é comum, tanto em enxames de abelhas, como em colónias de formigas, existirem elementos batedores, com comportamentos específicos, cujos resultados são posteriormente aproveitados pelo grosso da colónia/enxame.

Na forma em que as idealizámos, as partículas batedoras podem ser utilizadas tanto para introduzir aperfeiçoamentos no algoritmo global, por exemplo um sub-algoritmo de procura local, como para implementar mecanismos dependentes do problema. Estes permitem melhor adaptar o algoritmo para um problema específico, sem alterar todo o comportamento do enxame, já que não modificam directamente as regras de actualização gerais.

Uma vez que a interacção com o enxame principal é feita essencialmente através da partilha do melhor valor encontrado pelas partículas, a introdução de batedores, mesmo que o seu comportamento individual não seja produtivo, não vai perturbar substancialmente o comportamento global do enxame. Neste sentido, a utilização de partículas batedoras constitui uma forma muito flexível de introduzir novas capacidades no algoritmo de optimização, ou mesmo de o hibridizar com um optimizador completamente diferente.

Para ilustrar esta ideia, iremos utilizar três partículas batedoras para melhorar o desempenho do algoritmo predador-presa em problemas de optimização contínua. O funcionamento dessas partículas é descrito nos algoritmos 5.5, 5.6 e 5.7 e discutido nas próximas três subsecções. Na nossa implementação, estas partículas são actualizadas pelas suas próprias regras antes do ciclo geral de actualização do enxame. De um ponto de vista prático, uma partícula batedora pode ser, numa mesma iteração do enxame, actualizada pelas suas próprias regras e pela regra geral do enxame, embora frequentemente tal seja evitado, poupando assim avaliações da função objectivo.

Batedor que realiza procura local

A estratégia de procura escolhida para o primeiro batedor que vamos utilizar consiste na utilização de uma forma de procura local, que já sabemos ser uma forma comum de hibridização em muitos algoritmos de optimização evolucionários, sendo estes híbridos frequentemente chamados algoritmos meméticos. Seleccionamos para batedor a melhor partícula do enxame, já que é em torno desta que é mais promissor fazer uma procura local.

Algoritmo 5.5 Algoritmo para uma partícula batedora que realiza uma procura local.

```
...
...
...
para cada partícula i faz
    se f(p_i) < f(p_g) então
        g ← i
    fim se
fim para
d ← t mod m
x' ← p_g
x'_d ← x'_d + n(0, σ)
se f(x') < f(p_g) então
    p_g ← x'
    s ← s + 1
fim se
se ((t + 1) mod 100) = 0 então
    se s > 20 então
        σ ← σ × 2
    senão se s < 20 então
        σ ← σ ÷ 2
    fim se
    s ← 0
fim se
...
...
...
```

Em cada iteração do algoritmo é realizada uma mutação aleatória numa das dimensões de $\mathbf{p}_g$, p_{gd}, usando a equação (5.14), onde $n(0, \sigma)$ é um número aleatório com distribuição normal de média 0 e desvio padrão σ. O valor de $\mathbf{p}_g$ é actualizado para o novo $\mathbf{p}'_g$

somente se $f(\mathbf{p}'_g) < f(\mathbf{p}_g)$, i.e., se a posição resultante for melhor do que a anterior.

$$p'_{gd} = p_{gd} + n(0, \sigma) \tag{5.14}$$

O parâmetro σ tem como valor inicial $x_{max}/10$ e é actualizado durante a execução utilizando a regra do $1/5$ frequentemente usada nas estratégias evolutivas [Beyer and Schwefel, 2002]. Depois de cada 100 iterações, σ é dobrado se a taxa de sucesso da mutação for superior a $1/5$ e é reduzido para metade no caso contrário. Este mecanismo de mutação permite que uma procura local seja feita em torno de $\mathbf{p}_g$ ao longo do tempo. Note-se que, embora esta seja uma estratégia simples (mas mesmo assim capaz de bons resultados, como veremos), na prática poderia ser substituída por qualquer outra estratégia de procura local, de qualquer nível de sofisticação, sem alterar o restante do algoritmo.

Batedor que realiza procura por oposição

A segunda partícula batedora utiliza uma regra de actualização inspirada pela aprendizagem baseada em oposição[5] [Tizhoosh, 2005]. A ideia central por trás da ABO é que por vezes pode ser útil considerar, não apenas uma determinada posição no espaço de procura, mas também a posição oposta. Esta heurística mostrou-se inicialmente útil em tarefas de aprendizagem, acelerando algoritmos de aprendizagem por reforço e de treino de redes neuronais por retro-propagação do erro [Ventresca and Tizhoosh, 2006]. Posteriormente foram desenvolvidas extensões dos algoritmos de optimização por enxames de partículas [Wang et al, 2007] e de evolução diferencial [Rahnamayan et al, 2008], as quais mostraram melhorias significativas em relação aos algoritmos base.

Para esta partícula seleccionamos a partícula com pior avaliação, $\mathbf{p}_w$, com base na ideia de que a área oposta à posição actual da pior partícula pode constituir uma área interessante para explorar. Calculamos a posição oposta $\mathbf{p}'_w$ utilizando a equação (5.6) para cada dimensão d, sendo que h_d e l_d representam, respectivamente, o valor máximo e mínimo do enxame nessa dimensão (linhas 8-15 do algoritmo 5.6). Mais uma vez $\mathbf{p}_w$ é apenas actualizado para $\mathbf{p}'_w$ se $f(\mathbf{p}'_w) < f(\mathbf{p}_w)$.

$$p'_{wd} = h_d + l_d - u(0, 1)p_{wd} \tag{5.15}$$

[5] Do inglês *opposition based learning*.

Algoritmo 5.6 Algoritmo para uma partícula batedora que realiza uma procura baseada em oposição.

```
...
...
...
para cada partícula i faz
   se f(p_i) > f(p_w) então
      w ← i
   fim se
   para cada dimensão d faz
      se x_id > h_d então
         h_d ← x_id
      fim se
      se x_id < l_d então
         l_d ← x_id
      fim se
   fim para
fim para
para cada dimensão d faz
   x'_d ← u_d + l_d − u(0,1)p_wd
fim para
se f(x') < f(p_w) então
   p_w ← x'
fim se
se f(x') < f(p_g) então
   g ← w
fim se
...
...
...
```

Batedor que utiliza conhecimento específico sobre o problema

O terceiro batedor que iremos utilizar em algumas das experiências realizadas, em vez de utilizar mecanismos genéricos para melhorar o desempenho do optimizador, procura capitalizar o conhecimento específico sobre o problema para o fazer. Para simular esta situação, iremos deslocar o óptimo de algumas função de teste, de maneira a que em cerca de 80% das dimensões este seja 0, em cerca de 10% se encontre em x_{max}, e nos restantes 10% possa assumir qualquer valor do domínio.

O batedor aproveita este conhecimento deslocando uma diferente dimensão da partícula batedora, em cada iteração, para um dos extremos do domínio, tal como é descrito no algoritmo 5.7 (linhas 9-15). Para este batedor escolhemos a segunda melhor partícula do enxame (linhas 4-8 do algoritmo 5.7).

A utilização deste batedor cumpre um objectivo duplo. Por um lado, permite investigar se a adição de uma partícula batedora baseada em informação sobre o problema é viável, no sentido de produzir uma melhoria de desempenho que compense as avaliações adicionais da função a optimizar. Por outro lado, permite-nos simular as vantagens deste tipo de partícula para o nosso problema em particular, a optimização de máquinas de vector de suporte. Isto porque a situação sintética criada, nomeadamente a colocação de 80% do vector solução a 0, e apenas 10% de valores espalhados pelo domínio, mimetiza o tipo de soluções típicas das MVS.

Algoritmo 5.7 Algoritmo para uma partícula batedora específica para problemas com soluções nos extremos do espaço de procura.

```
1: ...
2: ...
3: ...
4: para cada partícula i faz
5:    se i ≠ g e f(p_i) < f(p_k) então
6:       k ← i
7:    fim se
8: fim para
9: d ← t mod m
10: x' ← p_k
11: se u(0,1) > 0.8 então
12:    x'_d ← x_max
13: senão
14:    x'_d ← 0
15: fim se
16: se f(x') < f(p_k) então
17:    p_k ← x'
18: fim se
19: se f(x') < f(p_g) então
20:    g ← k
21: fim se
22: ...
23: ...
24: ...
```

5.5 Evolução Diferencial

A evolução diferencial é outro algoritmo de optimização estocástico, baseado numa população de soluções, que partilha várias das características dos algoritmos de enxame, incluindo a simplicidade, popularidade e desempenho elevado [Das and Suganthan, 2011; Price et al, 2005; Vesterstrom and Thomsen, 2004]. O princípio central da evolução diferencial é que as novas posições no espaço de procura (ou indivíduos) são criados somando, a posições já existentes, a diferença entre outras duas posições presentes na população. Esta forma de geração dos novos indivíduos distingue claramente a evolução diferencial dos outros algoritmos evolucionários, além de fornecer o termo "diferencial" ao nome do algoritmo.

A primeira versão do algoritmo de evolução diferencial que utilizamos é chamada DE/$rand$/1/bin, uma formulação clássica descrita no algoritmo 5.8. Note-se que, à imagem do que acontece nos algoritmos de optimização por enxames de partículas, também na evolução diferencial existem inúmeras variantes, com diferentes forças e fraquezas [Das and Suganthan, 2011; Mezura-Montes et al, 2006].

No algoritmo apresentado, os indivíduos são inicializados aleatoriamente no domínio do problema. As linhas 5-14 do algoritmo 5.8 têm como objectivo seleccionar os três indivíduos necessários à criação de cada nova solução, enquanto na linha 15 é seleccionada aleatoriamente uma dimensão k, na qual o indivíduo corrente será obrigatoriamente modificado. Um novo indivíduo $\mathbf{x}_i$ é criado actualizando cada dimensão d da partícula i, x_{id}, através da equação (5.16), com probabilidade R. Caso a equação não seja utilizada, x_{id} mantém o valor correspondente do indivíduo anterior, p_{id} (linhas 16-23). O novo indivíduo substitui o anterior apenas se for melhor do que ele, i.e. se $f(\mathbf{x}_i) < f(\mathbf{p}_i)$.

$$x_{id} = p_{ad} + F(p_{bd} - p_{cd}) \tag{5.16}$$

O parâmetro R usado neste algoritmo age como uma probabilidade de recombinação, controlando a percentagem de valores de cada indivíduo $\mathbf{p}_i$ que são substituídos a cada geração. Já o parâmetro F da equação (5.16) é um parâmetro de escala que controla a intensidade das alterações, ao pesar a diferença que é adicionada aos valores base na criação de cada indivíduo. Valores típicos para estes parâmetros, são, respectivamente 0.9 e 0.5. Note-se que o desempenho deste algoritmo é notoriamente dependente da escolha destes parâmetros [Pedersen, 2010a], facto que constitui uma das razões pelas quais implementámos um segundo algoritmo baseado em evolução diferencial, este sem parâmetros de controlo.

Algoritmo 5.8 Algoritmo básico de evolução diferencial.

para cada partícula i **faz**
  $\mathbf{p}_i \leftarrow \mathbf{u}(x_{min}, x_{max})$
fim para
enquanto $t < t_{max}$ **e** $c < c_{max}$ **faz**
  para cada agente i **faz**
    repete
      $a \leftarrow \lfloor n \times u(0,1) \rfloor$
    até $a \neq i$
    repete
      $b \leftarrow \lfloor n \times u(0,1) \rfloor$
    até $b \neq i$ **e** $b \neq a$
    repete
      $c \leftarrow \lfloor n \times u(0,1) \rfloor$
    até $c \neq i$ **e** $c \neq a$ **e** $c \neq b$
    $k \leftarrow \lfloor m \times u(0,1) \rfloor$
    para cada dimensão d **faz**
      se $j = k$ **ou** $u(0,1) < R$ **então**
        $x_{id} \leftarrow p_{ad} + F(p_{bd} - p_{cd})$
      senão
        $x_{id} \leftarrow p_{id}$
      fim se
      $x_{id} \leftarrow \text{limita}(x_{id}, x_{min}, x_{max})$
    fim para
    se $f(\mathbf{x}_i) < f(\mathbf{p}_i)$ **então**
      $\mathbf{p}_i \leftarrow \mathbf{x}_i$
    fim se
  fim para
fim enquanto

5.6 Evolução Diferencial com Procura Livre

O algoritmo de evolução diferencial com procura livre (EDPL) procura ultrapassar a dependência do algoritmo base de uma correcta escolha dos seus parâmetros e, simultaneamente, melhorar o seu desempenho, hibridizando-o com um outro algoritmo de optimização estocástico baseado em populações - a chamada procura livre[6] (PL) [Penev and Littlefair, 2005]. A EDPL usa ainda uma modificação adicional baseada em aprendizagem por oposição, conceito que já descrevemos anteriormente a propósito do

[6]Do inglês *free search*.

algoritmo de optimização predador-presa com batedores.

Algoritmo 5.9 Evolução diferencial com procura livre.

para cada partícula i **faz**
  $\mathbf{p}_i \leftarrow \mathbf{x}_i \leftarrow \mathbf{u}(x_{min}, x_{max})$
  $g_i \leftarrow f(\mathbf{x}_i)$
fim para
enquanto $t < t_{max}$ **e** $c < c_{max}$ **faz**
  para cada indivíduo i **faz**
    se $u(0,1) < t/t_{max}$ **ou** $\max(\mathbf{g}) = \min(\mathbf{g})$ **então**
      $r \leftarrow n(0,1)$
      para cada dimensão d **faz**
        $x_{id} \leftarrow p_{id} + r$
        $x_{id} \leftarrow \text{limita}(x_{id}, x_{min}, x_{max})$
      fim para
    senão
      $s_i \leftarrow u(0,1)$
      repete
        $k \leftarrow \lfloor n \times u(0,1) \rfloor$
      até $\frac{g_k - \min(\mathbf{g})}{\max(\mathbf{g}) - \min(\mathbf{g})} > s_i$
      repete
        $a \leftarrow \lfloor n \times u(0,1) \rfloor$
        $b \leftarrow \lfloor n \times u(0,1) \rfloor$
      até $a \neq k$ **e** $b \neq k$ **e** $a \neq b$
      repete
        $w \leftarrow u(0,1)$
      até $w \neq 0$
      para cada dimensão d **faz**
        $x_{id} \leftarrow p_{kd} + (x_{ad} - x_{bd}) \ln(1/w)$
        $x_{id} \leftarrow \text{limita}(x_{id}, x_{min}, x_{max})$
      fim para
    fim se
    se $f(\mathbf{x}_i) < g_i$ **então**
      $\mathbf{p}_i \leftarrow \mathbf{x}_i$
      $g_i \leftarrow f(\mathbf{x}_i)$
    fim se
  fim para
fim enquanto

De forma muito resumida, a procura livre tentar imitar os padrões quotidianos de busca realizados pelos animais na natureza. Uma população de agentes, denominados

animais, inicia percursos aleatórios no espaço de procura, a partir de determinadas posições (que podem também elas ser aleatórias). Cada animal mantém em memória a melhor posição que encontrou e possui um sentido[7] que utiliza para escolher a posição de início do próximo percurso ou para se orientar no espaço de procura. Esta noção de sentido é o maior elemento de diferenciação da procura livre em relação a outros algoritmos de base populacional.

Como nos algoritmos anteriores, a EDPL inicializa as posições iniciais $\mathbf{x}_i$ dos indivíduos aleatoriamente no domínio do problema, guardando-as inicialmente como as melhores posições $\mathbf{p}_i$ encontradas até à data. Também avalia e armazena os resultados da avaliação g_i destas posições iniciais (linhas 1-4 do algoritmo 5.9).

No ciclo principal do algoritmo, os sentidos dos animais s_i são escolhidos aleatoriamente de uma distribuição uniforme entre 0 e 1 e as novas posições $\mathbf{x}_i$ a explorar são definidas utilizando a equação (5.17). Nesta, o primeiro caso corresponde à realização de uma busca local por parte do animal em torno da melhor posição que encontrou. O segundo caso, semelhante ao da ED clássica, corresponderá a uma procura global, sendo escolhida como atractor uma partícula k cuja avaliação normalizada seja melhor que o sentido do animal, isto é $\frac{g_k - \min(\mathbf{g})}{\max(\mathbf{g}) - \min(\mathbf{g})} > s_i$.

$$\mathbf{x_i} = \begin{cases} \mathbf{p}_i + n(0,1), \text{ se } u(0,1) < \frac{t}{t_{max}} \text{ ou } \max(\mathbf{g}) = \min(\mathbf{g}) \\ \mathbf{p}_k + (\mathbf{x}_a - \mathbf{x}_b)\ln(\frac{1}{u(0,1)}), \text{ senão.} \end{cases} \tag{5.17}$$

Na equação (5.17), a condição $u(0,1) < \frac{t}{t_{max}}$ garante que a procura global é favorecida no início da execução do algoritmo e ainda que, à medida que a iteração t avança, também a frequência da procura local aumenta. A nova posição $\mathbf{x}_i$ é armazenada em $\mathbf{p}_i$, se tiver uma melhor avaliação do que a posição em memória, i.e. se $f(\mathbf{x}_i) < g_i$. O processo de movimento das partículas é apresentado detalhadamente nas linhas 7-33 do algoritmo 5.9.

O último aspecto a considerar no algoritmo de evolução diferencial com procura livre tem a ver com a actualização do pior indivíduo, após o movimento de todos eles, utilizando aprendizagem por oposição, numa forma muito semelhante à por nós utilizada no OPPB e aqui descrita no algoritmo 5.10. A única diferença substancial em relação à nossa implementação reside no facto de, na EDPL, o cálculo da posição oposta ser feito em relação aos extremos do domínio do problema, x_{max} e x_{min} (linha 10 do algoritmo 5.10), enquanto nós o fazemos em relação aos extremos dos hiper-cubo ocupado pelo enxame na iteração em causa.

[7] Do inglês *sense*.

Algoritmo 5.10 Evolução diferencial com procura livre - procura por oposição.

```
...
...
...
para cada partícula i faz
    se f(p_i) > f(p_w) então
        w ← i
    fim se
fim para
para cada dimensão d faz
    x_wd = x_max + x_min − u(0, 1)p_wd
fim para
se f(x_w) < f(p_w) então
    p_w ← x_w
    g_w ← f(x_w)
fim se
...
...
...
```

5.7 Ambiente Experimental

Como é comum na área da optimização utilizando algoritmos evolucionários, os algoritmos atrás descritos foram testados empiricamente num ambiente experimental constituído por diversas funções de teste, seleccionadas especialmente para o efeito. As funções em questão foram escolhidas cuidadosamente da bibliografia, de maneira a colocar diversas dificuldades aos algoritmos, em correspondência com as questões que inicialmente identificámos, nomeadamente no que diz respeito à capacidade de lidar com múltiplos óptimos locais, elevada dimensionalidade, não-separabilidade e necessidade de rapidez de convergência. Antes de descrevermos o conjunto de funções de teste utilizado, vamos relembrar alguns termos que temos vindo a utilizar para descrever as funções a optimizar e que serão necessários para caracterizar as funções do conjunto de teste:

- Uma função é *unimodal* se tiver apenas um óptimo global. Para as funções unimodais é possível implementar algoritmos muito eficientes de optimização, no entanto continuam a ser utilizadas nos conjuntos de testes dos algoritmos evolucionários, porque permitem comparar as velocidades de convergências dos diversos algoritmos em zonas monotónicas de problemas mais complicados ou testar o seu comportamento quando enfrentam determinadas questões isoladamente,

como zonas planas ou quase planas do espaço de procura ou a não separabilidade da função.

- Uma função é multimodal se tiver dois ou mais óptimos locais. Esta será a situação na optimização de MVS com núcleos não positivos semi-definidos e já requer algoritmos capazes de optimização global, como os evolucionários, para evitar que a optimização fique presa num deste óptimos, não atingindo assim o óptimo global. Normalmente a função é mais fácil de optimizar se os óptimos estiverem organizados de uma forma regular, que o algoritmo possa explorar para se deslocar de óptimo em óptimo até à solução. O problema é mais complexo se a localização dos óptimos for aleatória no espaço de procura.

- Uma função é aditivamente separável se puder ser rescrita como uma soma de funções de uma variável. De uma forma mais lata, pode-se dizer que uma função é separável se puder ser optimizada para cada variável separadamente. As funções não separáveis, como a função objectivo nas máquinas de vectores de suporte, constituem problemas de optimização mais difíceis, já que a direcção de optimização mais eficiente está simultaneamente dependente de dois ou mais parâmetros da função.

Para analisar o desempenho do algoritmo proposto em relação a estes diferentes aspectos, seleccionámos 16 funções de teste da bibliografia da área da optimização em domínios contínuos, as quais dividimos em cinco grupos, de acordo com as características descritas anteriormente.

Tabela 5.1: Grupo 1 de funções de teste: funções unimodais.

$f_1(\mathbf{x}) = \sum_{i=1}^{n} x_i^2$
$f_2(\mathbf{x}) = \sum_{i=1}^{n} \left(\sum_{j=1}^{i} x_j \right)^2$
$f_3(\mathbf{x}) = \sum_{i=1}^{n} x_i^2 + \left(\sum_{i=1}^{n} 0.5 i x_i \right)^2 + \left(\sum_{i=1}^{n} 0.5 i x_i \right)^4$

No *grupo 1* encontram-se três funções unimodais, f_1-f_3, apresentadas na tabela 5.1. Estas três funções permitem-nos sobretudo perceber quão rapidamente um algoritmo converge numa zona monotónica do espaço de procura, mas adicionalmente, f_2 é não separável e, em f_3, o óptimo encontra-se numa zona muito plana do espaço de procura, com pouca informação do gradiente para guiar os algoritmos, o que nos permite perceber quais deles estagnarão nessa situação.

Tabela 5.2: Grupo 2 de funções de teste: funções multimodais com simetrias.

$f_4(\mathbf{x}) = 10n + \sum_{i=1}^{n} \left(x_i^2 - 10\cos(2\pi x_i)\right)$
$f_5(\mathbf{x}) = -20\exp\left(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}\right) - \exp\left(\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi x_i)\right) + 20 + e$
$f_6(\mathbf{x}) = \frac{1}{4000}\sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n}\frac{\cos(x_i)}{\sqrt{i}} + 1$
$f_7(\mathbf{x}) = 1 - \cos(2\pi\|\mathbf{x}\|) + 0.1\|\mathbf{x}\|$
$f_8(\mathbf{x}) = \sum_{i=1}^{n} \left(\lvert x_i\rvert^{0.8} + 5\sin(x_i^3)\right)$
$f_9(\mathbf{x}) = \sum_{i=1}^{n-1} s(x_i, x_{i+1}) + s(x_n, x_1), s(x,y) = 5 + \frac{\sin^2\left(\sqrt{x^2+y^2}\right) - 0.5}{\left(1 + 0.001(x^2+y^2)\right)^2}$
$f_{10}(\mathbf{x}) = 0.1\left(\sin^2(3\pi x_1) + \sum_{i=1}^{n-1}(x_i-1)^2\left(1+\sin^2(3\pi x_{i+1})\right) + (x_n-1)^2\left(1+\sin^2(2\pi x_n)\right)\right)$

As funções f_4-f_{10}, apresentadas na tabela 5.2, constituem o *grupo 2* de funções de teste. A característica comum a todas estas funções é o facto de serem funções multimodais com muitos óptimos locais e possuidoras de fortes simetrias em torno do óptimo global, localizado em $\mathbf{x_i^*} = 0$. Estas simetrias facilitam, até certo ponto, a tarefa aos algoritmos de optimização que as consigam explorar. Adicionalmente f_6, f_7, f_9 e f_{10} são funções não-separáveis. Para evitar que algoritmos com uma tendência embutida para convergir para o centro do referencial possam sair beneficiados nestes problemas, deslocámos o óptimo das funções f_1-f_{10} nos valores apresentados na tabela 5.4.

O *grupo 3* de funções, abrange as funções f_{11}-f_{16} e é apresentado na tabela 5.3. São também funções multimodais, mas não possuem, na generalidade, as simetrias presentes nas funções do grupo anterior. Apresentam diferentes obstáculos aos algoritmos de optimização, incluindo óptimos locais muito distanciados uns dos outros, óptimos globais perto dos limites do espaço de procura e/ou óptimos cercados por regiões muito ruidosas. Estas características tornam este grupo o que coloca mais desafios aos algoritmos, especialmente atendendo aos facto de entre estas funções apenas f_{12} e f_{14} serem separáveis.

O *grupo 4* e o *grupo 5* de funções de teste sobrepõem-se aos três grupos já descritos, agrupando respectivamente as funções separáveis e as funções não separáveis, de maneira a facilitar a referência a esses conjuntos de funções na análise dos resultados experimentais. Assim, o *grupo 4* é constituído por 7 funções, f_1, f_3-f_5, f_8, f_{12} e f_{14},

Tabela 5.3: Grupo 3 de funções de teste: funções multimodais sem simetrias.

$f_{11}(\mathbf{x}) = \sum_{i=1}^{n-1} \left(100\left(x_i^2 - x_{i+1}\right)^2 + (1 - x_i)^2\right)$
$f_{12}(\mathbf{x}) = -\sum_{i=1}^{n} \sin(x_i)\left(\sin\left(\frac{ix_i^2}{\pi}\right)\right)^{20}$
$f_{13}(\mathbf{x}) = \prod_{i=1}^{n}\left(\sum_{j=1}^{5} j\cos\left((j+1)x_i + j\right)\right)$
$f_{14}(\mathbf{x}) = 418.98291n + \sum_{i=1}^{n}\left(-x_i^2\sin(\sqrt{\lvert x_i\rvert})\right)$
$f_{15}(\mathbf{x}) = \frac{\sum_{i=1}^{n-1} r(x_i, x_{i+1}) + r(x_n, x_1)}{n},$ $r(x,y) = x\sin\left(\sqrt{\lvert y+1-x\rvert}\right)\cos\left(\sqrt{\lvert y+1+x\rvert}\right)$ $+ (y+1)\cos\left(\sqrt{\lvert y+1-x\rvert}\right)\sin\left(\sqrt{\lvert y+1+x\rvert}\right)$
$f_{16}(\mathbf{x}) = \sum_{i=1}^{n}\sum_{j=1}^{n}\left(\frac{\left(100(x_i^2 - x_j)^2 + (1-x_j)^2\right)^2}{4000} - \cos\left(100(x_i^2 - x_j)^2 + (1-x_j)^2\right) + 1\right)$

enquanto que o *grupo 5* é constituído pelas 9 funções restantes: f_2, f_6, f_7, f_9-f_{11}, f_{13}, f_{15} e f_{16}.

Refira-se ainda que todas as funções escolhidas são escaláveis no número de dimensões, o que permite a sua optimização com um número de dimensões arbitrária. Uma lista completa das funções utilizadas e dos seus parâmetros é apresentada na tabela 5.4, incluindo o nome mais comum pelo qual cada função é conhecida na bibliografia da área, o seu domínio de optimização, o valor óptimo $f(\mathbf{x}^*)$ e a sua posição $\mathbf{x}^*$ no espaço de procura (quando conhecidos). É também apresentado o valor de deslocação utilizado para as funções f_1-f_{10}.

Todas as funções são discutidas mais detalhadamente no Apêndice A. Além da descrição de cada função e da equação correspondente, são apresentadas representações gráficas das funções a duas dimensões, tanto na sua vista global, como, quando tal se justifica, incluindo uma vista da região em torno do óptimo. Procuramos desta forma ilustrar mais claramente as dificuldades que os algoritmos terão de ultrapassar para optimizar cada função.

Tabela 5.4: Parâmetros das funções de teste.

Função	Nome	Domínio	$f(\mathbf{x}^*)$	x_i^*	Deslocação
f_1	Esfera	$[-100 : 100]$	0	0	25
f_2	Elipsóide	$[-100 : 100]$	0	0	25
f_3	Zhakarov	$[-5 : 10]$	0	0	1.25
f_4	Rastrigin	$[-5.12 : 5.12]$	0	0	1.28
f_5	Ackley	$[-32 : 32]$	0	0	8
f_6	Griewangk	$[-600 : 600]$	0	0	150
f_7	Salomon	$[-100 : 100]$	0	0	25
f_8	Kursawe	$[-1000 : 1000]$	0	0	250
f_9	Shaffer	$[-100 : 100]$	0	0	25
f_{10}	Levy-Montalvo	$[-500 : 500]$	0	1	125
f_{11}	Rosenbrock	$[-5 : 10]$	0	1	0
f_{12}	Michalewicz	$[0 : \pi]$	na	π	0
f_{13}	Shubert	$[-10 : 10]$	na	na	0
f_{14}	Schwefel	$[-500 : 500]$	0	420.97	0
f_{15}	Rana	$[-520 : 520]$	-512.75	514.04	0
f_{16}	Whitley	$[-10 : 10]$	0	1	0

Capítulo 6

Resultados Experimentais dos Optimizadores

Neste capítulo apresentamos e discutimos os resultados obtidos através da aplicação dos algoritmos descritos no capítulo anterior ao conjunto de funções de teste. Estes resultados estão separados em três secções diferentes. Na primeira secção apresentamos os resultados globais que permitem comparar o nosso algoritmo final, o algoritmo de optimização predador-presa com batedores, com os restantes algoritmos. Na segunda secção apresentamos os resultados intermédios, obtidos após a introdução de cada novo mecanismo no OPPB. Este conjunto de experiências permite ilustrar o papel das diversas partículas no algoritmo final, bem como a contribuição de cada componente para o desempenho global do algoritmo. Finalmente, na terceira secção, alteramos ligeiramente alguns dos problemas e utilizamos o conhecimento sobre essas alterações para introduzir um novo batedor. Os resultados permitem, por um lado, avaliar o desempenho dos algoritmos nos problemas modificados e, por outro lado, demonstrar a facilidade de introdução no algoritmo de partículas com regras de utilização baseadas em conhecimento sobre o problema, bem como a potencialidade destas para melhorar o desempenho do OPPB.

Todas as experiências foram realizadas com as funções definidas para 40 dimensões, sendo cada algoritmo executado 50 vezes para cada função. O gerador de números aleatórios foi inicializado de maneira a garantir que todos os algoritmos começam com a mesma população em execuções correspondentes. Considerou-se condição de paragem de cada execução a realização de $2e5$ avaliações da função objectivo. Após o término da execução, cada algoritmo devolve o melhor valor obtido para a função objectivo. A métrica escolhida para a avaliação dos algoritmos foi o valor médio e o desvio padrão do valor, devolvidos após as 50 execuções de cada algoritmo para cada função. Esta métrica foi preferida à outra métrica comum em problemas de optimização, o número médio de avaliações da função objectivo necessários para o algoritmo atingir

uma vizinhança δ do óptimo global, por três razões principais:

1. Para alguns dos problemas considerados não é conhecido o óptimo global, logo é impossível calcular quando δ foi atingido.
2. A utilização de um δ igual para todos os problemas é algo problemática, já que enquanto para alguns problemas pode ser trivial atingir essa diferença em relação ao óptimo, para outros pode ser quase impossível.
3. Ao considerarmos a métrica do valor médio obtido após um valor fixo de avaliações em associação com os gráficos de convergência que também apresentamos, é possível não só ter informação sobre o desempenho do algoritmo no fim da execução, mas, analisando os gráficos, podemos também saber quando é que qualquer vizinhança do óptimo foi atingida.

O tamanho dos enxames/populações foi limitado a 20 para todos os algoritmos. Os restantes parâmetros dos algoritmos assumiram os valores recomendados pelos autores, já mencionados aquando da descrição dos algoritmos e que sumariamos de seguida:

- OEP - $\phi_1 = \phi_2 = 2.0$, w varia linearmente entre 0.9 e 0.4.
- OEPHPMA - m=10, k=40, α varia linearmente entre 0.45 e 0.35.
- OPPB - $\phi_1 = \phi_2 = 1.6$, w varia linearmente entre 0.4 e 0.2, $r = 0.0008$.
- ED - $C = 0.9, F = 0.5$.
- EDPL - não possui parâmetros controláveis.

Como nota adicional, convém mencionar que os algoritmos OEPHPMA e EDPL foram inicialmente testados nas condições experimentais descritas pelos respectivos autores nos artigos onde são apresentados, com o objectivo de garantir que não existiam disparidades de implementação que invalidassem os resultados.

6.1 Resultados Globais

Na tabela 6.1 apresentamos os resultados obtidos pelos algoritmos descritos nas 16 funções de teste. Observando os valores obtidos, podemos concluir que, numa perspectiva global, o algoritmo de optimização predador-presa obteve os melhores resultados em 12 das 16 funções de teste, o que estabelece a sua competitividade com os restantes algoritmos utilizados na comparação. Este resultado é ainda mais significativo por o

segundo melhor algoritmo, o EDPL, ter obtido o melhor resultado em apenas 3 das funções de teste. Já o OEP híbrido apenas foi melhor que os restantes algoritmos numa das funções de teste.

As versões híbridas dos algoritmos tiveram um melhor desempenho do que as versões básicas na maioria das funções de teste, como aliás, seria de esperar. De facto, o OEPHPMA apenas não foi superior ao OEP em três funções. Já o algoritmo de evolução diferencial com procura livre é inferior à versão de ED básica em 7 dos problemas de teste, o que revela um desempenho pior do que o esperado, tendo em atenção os resultados publicados anteriormente. A justificação mais plausível para estes resultados, sendo que a diferença mais significativa no ambiente experimental em relação aos resultados originais consiste na deslocação do óptimo em relação à origem nas funções afectadas, parece-nos estar relacionada com uma tendência embutida no algoritmo para procurar a origem do referencial.

Analisando os resultados para cada um dos grupos de funções que definimos no capítulo anterior, verifica-se que para as funções do grupo 1, funções unimodais, o nosso algoritmo é claramente o melhor. Este resultado revela uma rapidez de convergência acentuada e a capacidade do algoritmo lidar com zonas planas do espaço de procura e, pelo menos no contexto das funções multimodais, com a não separabilidade do problema. Note-se que nestas funções a EDPL tem um comportamento claramente medíocre, o que indica que provavelmente estará a trocar alguma velocidade de convergência por diversidade adicional, a qual lhe será útil noutros problemas. A desvantagem de ser um algoritmo sem parâmetros revela-se nestas situações, já que torna impossível afinar o algoritmo para as especificidades de um problema em particular.

Em relação ao grupo 2, funções multimodais com fortes simetrias, o OPPB consegue obter os melhores resultados em 5 das 7 funções, o que indica que possui a capacidade de fugir aos óptimos locais pelo menos quando a sua organização é razoavelmente regular. O algoritmo de EDPL e o OEPHPMA obtiveram ambos o melhor resultado numa das funções restantes. O grupo 3 é constituído por funções multimodais mais heterogéneas e, também neste o OPPB tem o melhor desempenho global, obtendo os melhores resultados em 4 das 6 funções, sendo que nas restantes duas o algoritmo de EDPL é o melhor.

Analisando os resultados em função da separabilidade dos problemas, sendo que as funções separáveis estão agrupadas no grupo 4 e as não separáveis agrupadas no grupo 5, temos que o OPPB tem os melhores resultados em todas as funções do grupo 4 e em 5 funções do grupo 5. Embora no grupo de funções não-separáveis a supremacia do OPPB não se revele tão acentuada como nos grupos anteriores, o desempenho do algoritmo continua a ser substancialmente melhor do que o segundo melhor, DELP, o qual obteve os melhores resultados em 3 funções. Na função restante, o OEPHPMA foi o algoritmo que obteve melhor resultado.

Tabela 6.1: Resultados experimentais globais: média e desvio padrão dos melhores valores encontrados para 50 execuções de cada algoritmo.

	OEP	ED	OEPHPMA	EDPL	OPPB
f_1	1.65169e-26 (5.54796e-26)	6.25047e-23 (4.4143e-22)	3.38079e-23 (1.4217e-23)	0.242096 (0.174053)	**7.57306e-31** **(3.02794e-30)**
f_2	316.126 (254.974)	56.2572 (397.747)	0.00226426 (0.00212582)	61.5534 (33.1706)	**2.52261e-09** **(2.40497e-09)**
f_3	15.8846 (9.53778)	0.000692142 (0.0016271)	5.19511e-09 (4.86953e-09)	1.13648 (0.807662)	**6.72887e-10** **(5.23577e-10**)
f_4	55.8172 (17.0791)	148.852 (84.0428)	17.8104 (5.6274)	7.72839 (11.3974)	**0.0198992** **(0.140708)**
f_5	0.476326 (2.91565)	5.47737 (6.9252)	1.27917e-12 (2.26036e-13)	0.150284 (0.0844456)	**3.96305e-14** **(8.27882e-15)**
f_6	0.0134364 (0.0169437)	0.0861482 (0.282405)	**0.00777161** **(0.0112705)**	0.290238 (0.191083)	0.0524945 (0.0498488)
f_7	0.639873 (0.137024)	0.389874 (0.194044)	0.553874 (0.0973316)	**0.375873** **(0.169706)**	1.19387 (0.219842)
f_8	-41.2553 (50.7622)	-76.2049 (67.5841)	-24.9186 (25.9715)	171.819 (215.74)	**-150.326** **(16.6553)**
f_9	5.15659 (2.34525)	15.976 (0.391461)	6.77497 (1.32032)	12.0213 (1.81459)	**0.860774** **(0.424196)**
f_{10}	0.0181309 (0.126894)	0.0621908 (0.216245)	5.56247e-22 (3.83363e-22)	55.1298 (78.7154)	**5.86288e-28** **(3.38157e-28)**
f_{11}	56.518 (37.0099)	41.2635 (42.3479)	37.1949 (28.3189)	**0.0592396** **(0.0552312)**	4.31659 (13.8899)
f_{12}	-33.667 (1.29863)	-16.4953 (0.953367)	-36.3654 (1.52178)	-23.9191 (2.08068)	**-36.9508** **(1.04873)**
f_{13}	-6.50704e+44 (8.53809e+44)	-3.24599e+31 (1.58113e+32)	-5.44541e+45 (3.81935e+45)	-3.48103e+45 (7.0238e+45)	**-7.44425e+45** **(2.21794e+45)**
f_{14}	3206.31 (571.308)	4523.58 (1102.55)	579.052 (581.897)	2317.55 (1343.43)	**71.0635** **(92.6734)**
f_{15}	-325.619 (15.4024)	-466.38 (19.8534)	-401.814 (27.1564)	-417.29 (51.9677)	**-475.215** **(24.1802)**
f_{16}	357.564 (248.078)	404.922 (264.003)	181.022 (196.041)	**3.3508** **(18.8189)**	44.7104 (68.9411)

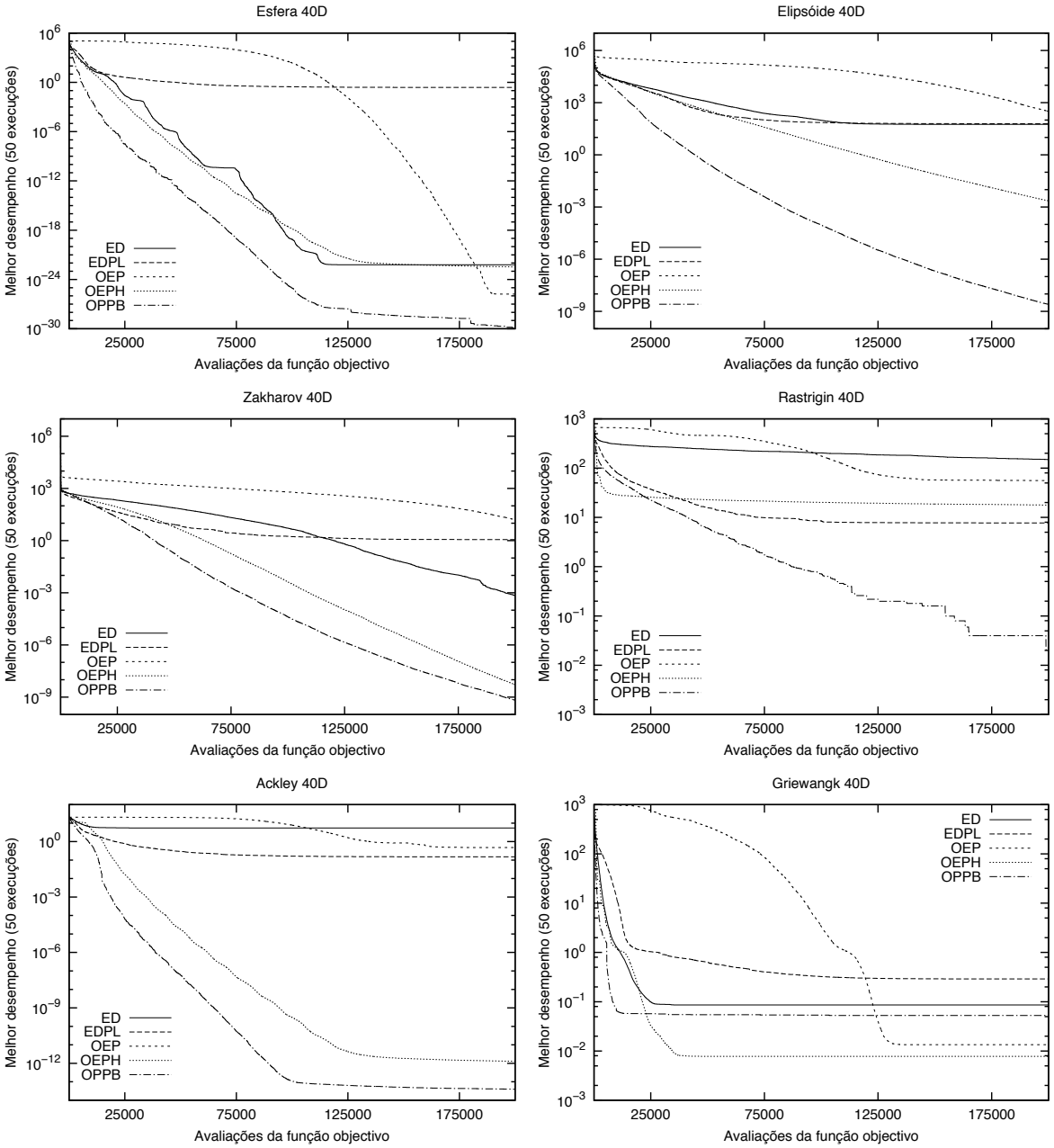


Figura 6.1: Gráficos de convergência para as funções f_1-f_6 (resultados globais).

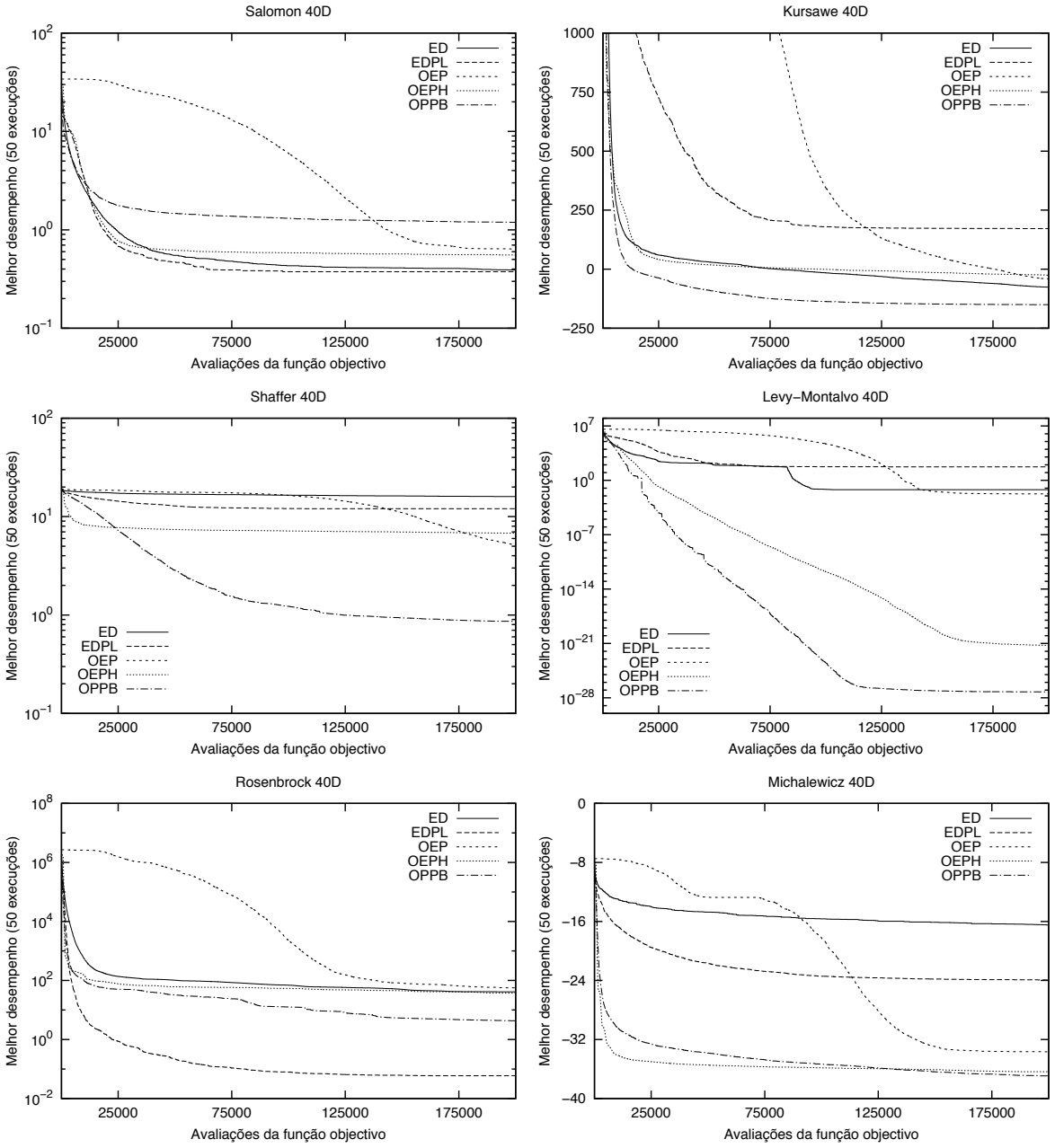


Figura 6.2: Gráficos de convergência para as funções f_7-f_{12} (resultados globais).

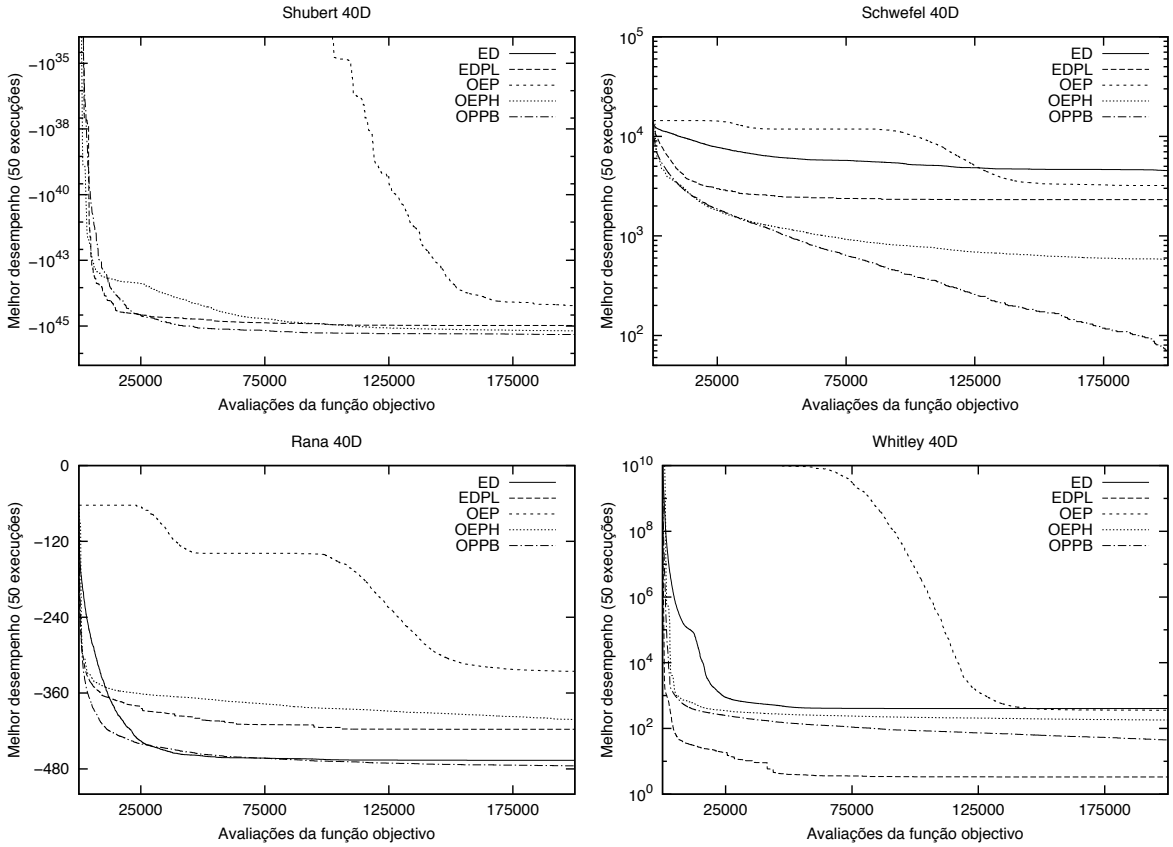


Figura 6.3: Gráficos de convergência para as funções f_{13}-f_{16} (resultados globais).

Nas figuras 6.1, 6.2 e 6.3 apresentamos gráficos de convergência para todas as funções de teste e todos os algoritmos. Cada gráfico corresponde a uma função e neles são apresentadas curvas para cada algoritmo, registando o progresso do melhor desempenho médio nas 50 execuções (eixo vertical) ao longo das $2e5$ avaliações permitidas para a função objectivo (eixo horizontal). Geralmente a escala do eixo vertical é logarítmica (notação 10^n no eixo) embora, quando tal não é possível, ou torne o gráfico pouco claro, a escala linear seja utilizada. As curvas estão associadas às siglas utilizadas ao longo deste documento para identificar os algoritmo, exceptuando OEPHPMA que é abreviado para OEPH.

Estes gráficos permitem observar de forma mais detalhada o desempenho dos algoritmos ao longo do tempo, sendo de sublinhar a rapidez de convergência que o OPPB apresenta na generalidade dos problemas, ao mesmo tempo que evita a convergência prematura para um óptimo local. É também possível perceber quantas avaliações em média um algoritmo necessita para chegar a uma determinada vizinhança δ do óptimo global.

6.2 Resultados Intermédios

Na tabela 6.2 apresentamos os resultados parcelares obtidos pelo novo algoritmo à medida que vão sendo acrescentados novos mecanismos. A primeira coluna corresponde aos resultados obtidos pelo OEP básico para as 16 funções. A segunda coluna apresenta os resultados para o algoritmo de optimização predador-presa (OPP) sem partículas batedoras. Na terceira coluna apresentamos os resultados para o algoritmo predador-presa com a adição da partícula batedora que realiza procura baseada em oposição (OPPO). Os resultados obtidos para o OPP em conjunção com a partícula que realiza procura local (OPPL) são apresentados na quarta coluna. A última coluna apresenta os resultados para o OPPB completo. A análise destes resultados permite-nos perceber qual a contribuição de cada componente do algoritmo para o seu desempenho final.

Comparando os resultados do OPP, o algoritmo com o mecanismo predador-presa e sem batedores, com o algoritmo de OEP original, podemos verificar que há uma melhoria substancial em 11 das 16 funções. Este efeito é mais substancial nas funções do grupo 3 (melhoria de 6 em 6), menor nas funções do grupo 2 (4 em 7), enquanto que nas funções unimodais apenas há melhoria em uma das 3 funções, tendo os resultados inclusivamente piorado nas outras duas. Estes resultados sugerem que o efeito predador-presa é sobretudo importante nos problemas mais complexos, onde a manutenção da diversidade é importante, mas que pode ter algum custo em termos de velocidade de convergência do algoritmo.

A adição do batedor que realiza procura por oposição traz melhorias em 8 das funções de teste, embora essa melhoria só seja substancial na função f_5 e, em menor escala, f_6 e f_{11}. Como os resultados de optimização de algumas funções também pioram com esta partícula, é claramente necessário algum cuidado na sua utilização. Estes resultados, particularmente em f_5, ilustram a utilização de uma partícula cujo comportamento é particularmente adequado à optimização de um problema específico.

Os resultados utilizando a partícula de procura local são substancialmente diferentes dos do outro batedor. Logo nas funções do grupo 1 nota-se um incremento muito substancial da velocidade de convergência, com resultados finais igualmente melhores nas três funções unimodais. Além destas, há ainda melhoria em 7 das restantes funções para um total de 10 em 16. Nas funções que não melhoram não há, ainda assim, um decréscimo significativo do desempenho.

O efeito desta partícula é, portanto, bastante mais distribuído do que para a anterior, havendo melhorias substanciais em muita das funções de teste. Note-se ainda o aumento da velocidade de convergência nas funções unimodais, o qual sugere que este batedor tem um efeito complementar em relação ao mecanismo-predador presa. Assim, a partícula com procura local ilustra a adição de um batedor que melhora o algoritmo

na generalidade. O efeito combinado destes mecanismos, no algoritmo OPPB, produz resultados que são superiores em 14 funções aos do OEP básico, em muitas delas por várias ordens de magnitude.

Tabela 6.2: Resultados experimentais intermédios: média e desvio padrão dos melhores valores encontrados para 50 execuções de cada algoritmo.

	OEP	OPP	OPPO	OPPL	OPPB
f_1	1.65169e-26 (5.54796e-26)	**1.71656e-28** **(3.5187e-28)**	1.63629e-27 (5.77022e-27)	**0** **(0)**	**7.57306e-31** **(3.02794e-30)**
f_2	316.126 (254.974)	1260.18 (2219.77)	1712.43 (2802.44)	**3.9881e-10** **(4.35113e-10)**	**2.52261e-09** **(2.40497e-09)**
f_3	15.8846 (9.53778)	31.3115 (46.2811)	**24.5038** **(40.7141)**	**4.62151e-10** **(5.1526e-10)**	**6.72887e-10** **(5.23577e-10**
f_4	55.8172 (17.0791)	**0.139295** **(0.853063)**	**0.0397984** **(0.196951)**	0.23879 (1.24822)	**0.0198992** **(0.140708)**
f_5	0.476326 (2.91565)	2.29725 (6.23978)	**7.24576e-14** **(2.50212e-14)**	**2.01927** **(5.62937)**	**3.96305e-14** **(8.27882e-15)**
f_6	0.0134364 (0.0169437)	0.139295 (0.853063)	**0.0624817** **(0.0569936)**	**0.0535327** **(0.0535873)**	**0.0524945** **(0.0498488)**
f_7	0.639873 (0.137024)	1.21987 (0.192725)	1.22987 (0.208248)	1.15387 (0.183181)	**1.19387** **(0.219842)**
f_8	-41.2553 (50.7622)	**-153.179** **(5.3920)**	**-153.365** **(2.16092)**	-150.644 (1.26064)	-150.326 (16.6553)
f_9	5.15659 (2.34525)	**0.855093** **(0.420195)**	0.920654 (0.434841)	**0.797715** **(0.413331)**	0.860774 (0.424196)
f_{10}	0.0181309 (0.126894)	**5.41382e-27** **(2.00007e-26)**	2.0807e-26 (5.06554e-26)	**4.46798e-28** **(3.52975e-28)**	**5.86288e-28** **(3.38157e-28)**
f_{11}	56.518 (37.0099)	**8.05825** **(17.0834)**	**3.1014** **(10.6769)**	**0.87747** **(2.87142)**	**4.31659** **(13.8899)**
f_{12}	-33.667 (1.29863)	**-37.5854** **(0.977789)**	-37.3926 (1.00457)	-37.4659 (1.0435)	-36.9508 (1.04873)
f_{13}	-6.50704e+44 (8.53809e+44)	**-8.19358e+45** **(2.15179e+45)**	**-8.19565e+45** **(2.82452e+45)**	-7.57977e+45 (2.98912e+45)	-7.44425e+45 (2.21794e+45)
f_{14}	3206.31 (571.308)	**49.7446** **(89.8183)**	54.4821 (80.1147)	49.7446 (83.1998)	71.0635 (92.6734)
f_{15}	-325.619 (15.4024)	**-462.301** **(15.8498)**	**-466.91** **(12.7634)**	**-467.015** **(13.2081)**	**-475.215** **(24.1802)**
f_{16}	357.564 (248.078)	**56.5527** **(124.553)**	60.2484 (89.5563)	**49.4296** **(69.9519)**	**44.7104** **(68.9411)**

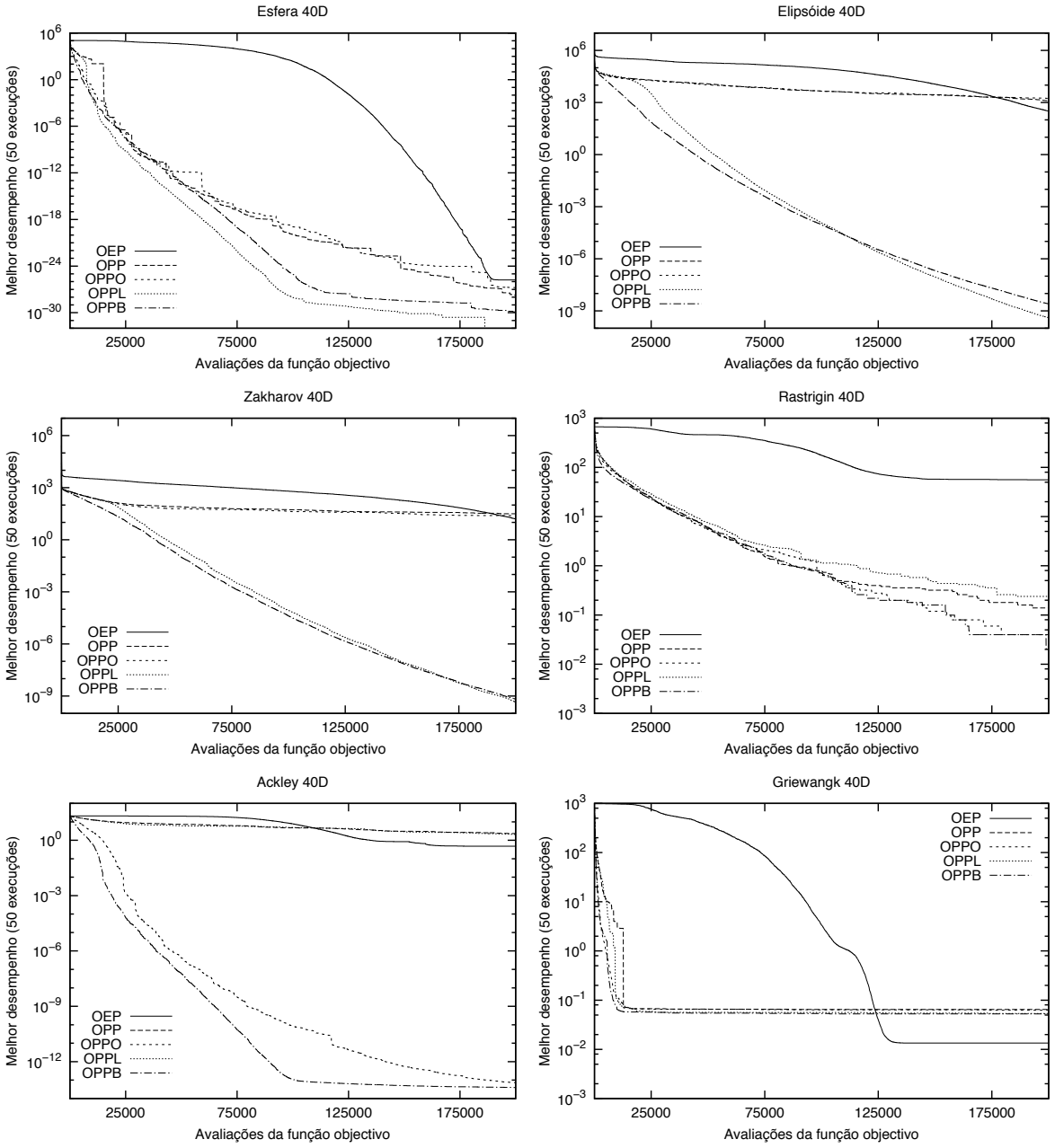


Figura 6.4: Gráficos de convergência para as funções f_1-f_6 (resultados intermédios).

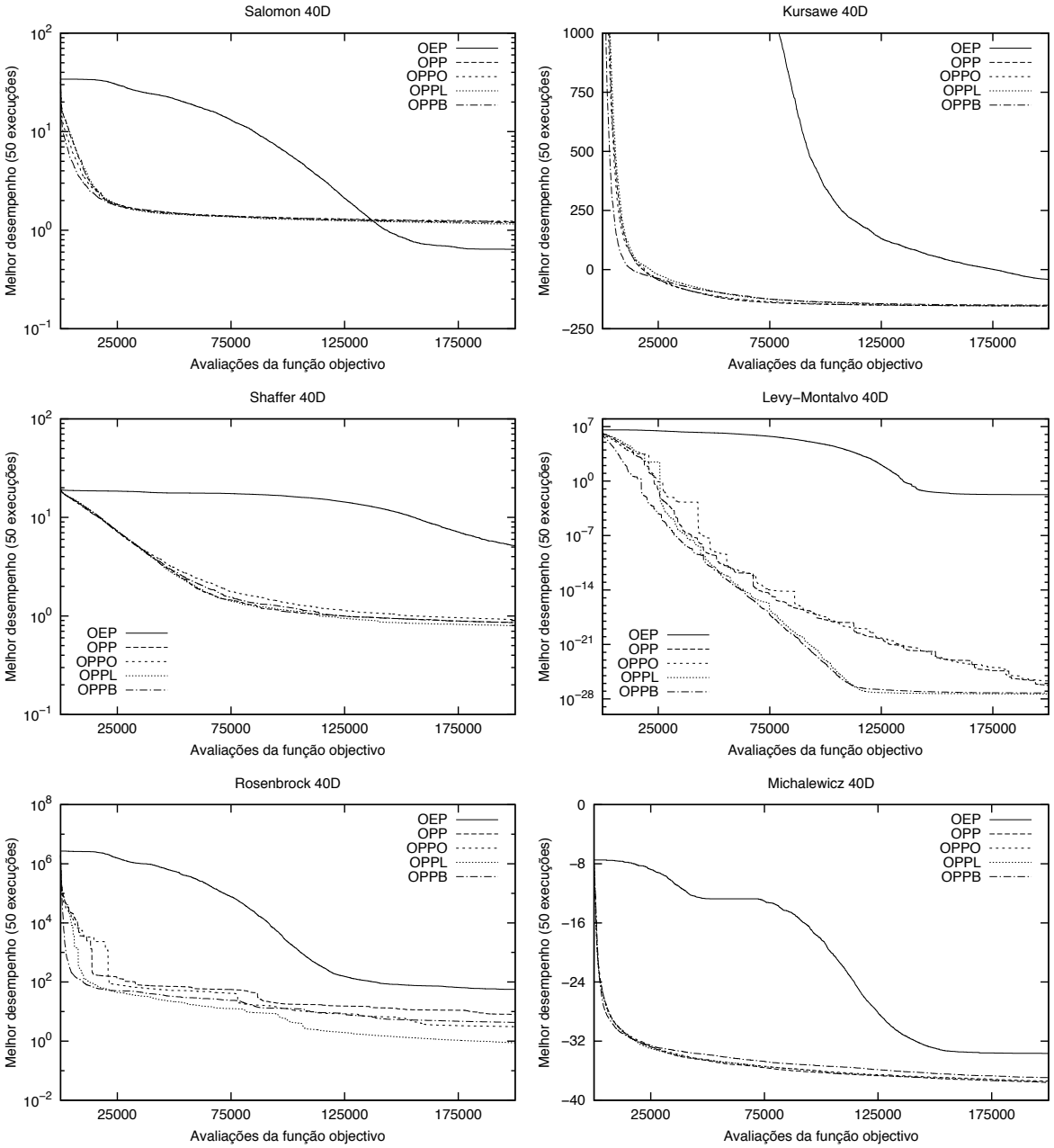


Figura 6.5: Gráficos de convergência para as funções f_7-f_{12} (resultados intermédios).

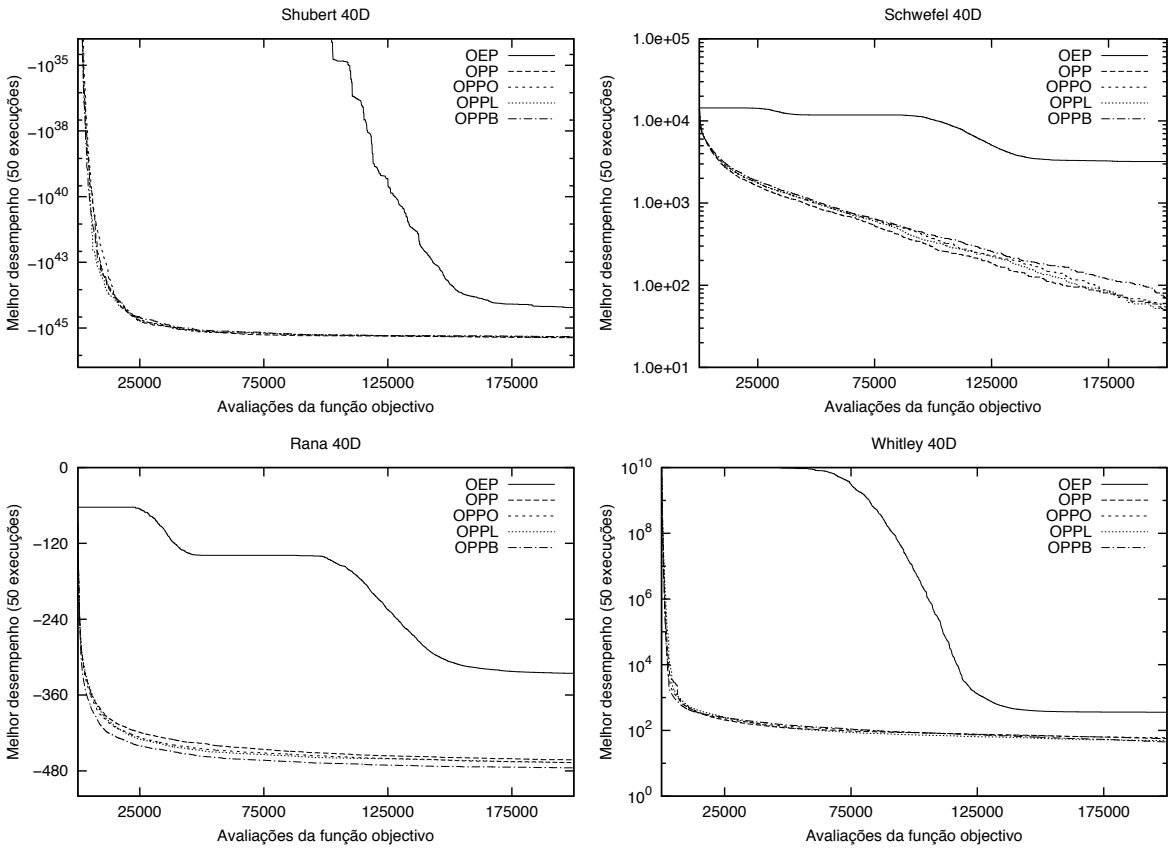


Figura 6.6: Gráficos de convergência para as funções f_{13}-f_{16} (resultados intermédios).

Quando comparamos apenas com o OPP, verifica-se que a adição das partículas batedoras melhora os resultados obtidos em 12 das 16 funções de teste, sendo que apenas na função f_{14} se verifica um decréscimo de desempenho digno de menção. Estabelecemos assim que as partículas batedoras podem ser adicionadas cumulativamente ao algoritmo, combinando os ganhos respectivos, sem acrescentar custos significativos para o desempenho global.

Nas figuras 6.4, 6.5 e 6.6 apresentamos gráficos de convergência para todas as funções de teste e todos os algoritmos, neste novo conjunto de experiências. Além do nível de detalhe sobre o desempenho dos algoritmos que se pode retirar destes gráficos, e que já descrevemos para o conjunto de experiências anterior, neste caso podemos ainda usá-los para ter uma ideia de qual dos novos mecanismos mais contribui para o desempenho do algoritmo final. Quando as quatro curvas respeitantes aos algoritmos com efeito predador-presa andam bastante juntas, é este o mecanismo dominante. Se a curva do algoritmo final (OPPB) é muito semelhante a uma das curvas correspondente a um algoritmo com uma das partículas batedoras, é porque é o contributo desta que está a definir o comportamento do OPPB.

6.3 Resultados Utilizando um Terceiro Batedor

Nesta secção procurámos testar todos algoritmos numa situação que de alguma forma imita o posicionamento dos óptimos locais nas máquinas de vectores de suporte. Para tal utilizámos apenas as funções dos grupos 1 e 2, já que devido à sua simetria e posicionamento do óptimo global no centro do referencial, nos permitem facilmente deslocar esse óptimo para qualquer posição do espaço de procura. Para simular o que acontece com os parâmetros α associados aos vectores de suporte, deslocámos os óptimos nas funções de teste de forma a que em 80% das dimensões se encontrassem no valor 0, e que, nos valores restantes, 80% fossem x_{max}. Os domínios das funções foram ainda ajustados para os intervalos $[0, x_{max}]$. Um novo óptimo foi criado para cada função e cada execução do algoritmo. Note-se que, desta forma, os óptimos das funções ficam nos limites do domínio em quase todas as dimensões, o que pode causar dificuldades significativas a alguns dos algoritmos.

Tabela 6.3: Resultados experimentais para as funções com óptimos deslocados: média e desvio padrão dos melhores valores encontrados para 50 execuções de cada algoritmo.

	OEP	ED	OEPHPMA	EDPL	OPPB
f_1	4.71783e-27 (1.20604e-26)	100 (707.107)	7.3948e-26 (2.60705e-25)	1.04446 (7.38415)	**0** **(0)**
f_2	94.8632 (60.3202)	1.05275 (7.44404)	3372.4 (10336.1)	13183.7 (9154.16)	**6.95768e-29** **(3.01847e-28)**
f_3	0.771589 (0.587236)	4.81932 (21.4369)	31.8958 (225.537)	158.062 (102.318)	**1.07958e-28** **(5.73612e-28)**
f_4	53.5899 (12.8791)	46.7979 (35.6009)	**0** **(0)**	40.1661 (37.248)	0.0198992 (0.140708)
f_5	4.94023e-13 (3.27447e-12)	1.42069 (4.25043)	2.25064e-14 (2.2529e-14)	15.0721 (4.13056)	**0** **(0)**
f_6	**0.0106737** **(0.0116833)**	0.0284011 (0.129813)	0.0322881 (0.0425845)	0.648994 (1.87741)	0.0450732 (0.062597)
f_7	0.637873 (0.0923392)	**0.221873** **(0.050669)**	0.405873 (0.130008)	1.51993 (2.52228)	0.335876 (0.158765)
f_8	-44.2543 (41.5538)	24.4846 (129.747)	-39.5094 (11.0244)	751.95 (395.148)	**-55.8945** **(10.2041)**
f_9	6.79306 (2.87295)	5.66485 (2.06892)	2.70569 (1.11255)	7.32397 (2.4941)	**0.0223451** **(0.0842491)**
f_{10}	3.69801 (26.1484)	2000.03 (6851.18)	5.19578e-06 (1.98551e-05)	173886 (239596)	**3.77984e-27** **(4.94146e-27)**

As experiências cujos resultados estão resumidos na tabela 6.3 mostram realmente

resultados bastantes piores para o algoritmo EDPL, que não consegue os melhores resultados em nenhuma função e encontra inclusivamente grandes dificuldades para optimizar algumas delas. Estas dificuldades notam-se especialmente nas três funções do grupo 1, mas também em várias das funções restantes, como se pode constatar na tabela 6.3. O desempenho do OEPHPMA decaiu mais graciosamente, e o algoritmo consegue inclusivamente obter os melhores resultados para a função f_4. No entanto também apresenta muitas dificuldades na optimização de funções como f_2, f_3, f_9 e f_{10}. Os resultados dos algoritmos de base também não são muito bons, embora cada um consiga obter os melhores resultados numa função do conjunto de teste.

Finalmente, no que diz respeito ao algoritmo proposto por nós, podemos observar que o OPPB foi o único algoritmo que não apresentou um decréscimo substancial, em termos de desempenho de optimização, para nenhuma das funções consideradas. Inclusivamente o seu desempenho melhorou em várias das funções, apesar dos óptimos estarem em posições menos acessíveis do espaço de procura e de serem diferentes em cada instanciação do problema. Em resumo, o OPPB obteve os melhores resultados em 7 das 10 funções de teste, sendo de referir mais uma vez que nenhum dos algoritmos restantes conseguiu o melhor desempenho em mais do que uma função. Estes resultados sugerem que o OPPB se encontra bem preparado para encontrar óptimos situados, na maioria das dimensões, nas fronteiras do espaço de procura, qualidade que esperamos se revele útil na optimização das máquinas de vectores de suporte.

O último conjunto de experiências realizadas tem como objectivo demonstrar as potencialidades das partículas batedoras como mecanismo de introdução de conhecimento sobre o problema no algoritmo. Para o conseguir, introduzimos uma terceira partícula batedora no OPPB, descrita no algoritmo 5.7, a qual utiliza o conhecimento sobre as alterações na localização dos óptimos que descrevemos anteriormente, numa tentativa de melhorar o desempenho do OPPB neste contexto. De forma resumida, lembramos que este batedor desloca uma diferente dimensão da partícula, em cada iteração, para um dos extremos do domínio, esperando assim aproveitar o facto de sabermos que, na maioria das dimensões, ser aí que a solução se encontra.

As experiências, cujos resultados apresentamos na tabela 6.4, foram feitas utilizando as funções e alterações descritas para o conjunto de experiências anterior, mas apenas para o algoritmo OPPB e a uma versão deste aumentada pela terceira partícula batedora. Esta versão do algoritmo aparece referida na tabela como OPPB+.

Os resultados obtidos, nos quais o OPPB+ obteve melhores resultados em 8 das 10 funções, permitem-nos confirmar a possibilidade de utilização de partículas batedoras como um mecanismo bastante simples para melhorar o desempenho de um algoritmo num problema específico, encorporando, para tal, conhecimento existente *a priori* sobre esse problema. Embora este exemplo seja bastante simples, ilustra no entanto as vantagens de utilização de partículas batedoras para este efeito: não é necessária a al-

teração generalizada do algoritmo, apenas uma pequena alteração no comportamento de uma partícula isolada, de maneira que o novo comportamento não perturbou o funcionamento geral do enxame, nem levou a um desperdício de avaliações da função objectivo que anulasse o efeito positivo introduzido pela nova partícula.

Tabela 6.4: Resultados experimentais com adição de um terceiro batedor: média e desvio padrão dos melhores valores encontrados para 50 execuções de cada algoritmo.

	OPPB	OPPB+
f_1	0 [9200]	**0 [7250]**
f_2	6.95768e-29 (3.01847e-28)	**3.75725e-29 (2.04331e-28)**
f_3	1.07958e-28 (5.73612e-28)	**4.05921e-29 (1.33317e-28)**
f_4	0.0198992 (0.140708)	**0 (0)**
f_5	0 [16900]	**0 [15150]**
f_6	**0.0450732 (0.062597)**	0.0471287 (0.0583248)
f_7	0.335876 (0.158765)	**0.293873 (0.142012)**
f_8	**-55.8945 (10.2041)**	-53.3964 (9.4122)
f_9	0.0223451 (0.0842491)	**0.000777273 (0.0038465)**
f_{10}	3.77984e-27 (4.94146e-27)	**3.87103e-28 (1.38232e-27)**

6.4 Conclusões

Neste capítulo testámos experimentalmente um novo algoritmo de optimização baseado em partículas de enxames. O algoritmo foi desenvolvido tendo em vista não só o desempenho em problemas de optimização genéricos, mas também a capacidade para lidar com dificuldades que esperamos encontrar na optimização de máquinas de vectores de suporte. Para testar o seu desempenho comparámo-lo extensivamente, tanto com os algoritmos básicos de dois paradigmas de optimização diferentes (inteligência de enxame e evolução diferencial), como com variantes estado da arte dos mesmos algoritmos. Essas comparações foram feitas utilizando um conjunto de funções de teste cuidadosamente seleccionadas para ilustrar os pontos fortes e fracos dos algoritmos em comparação.

De um ponto de vista geral, sobre o conjunto total das experiências realizadas, o algoritmo de optimização predador-presa com batedores, revelou-se o que melhor desempenho obteve, no sentido em que, utilizando a métrica de comparação que definimos, foi o que obteve sempre os melhores resultados de optimização para uma maioria substancial das funções em cada conjunto de experiências. Dos algoritmos em comparação, foi

também o que revelou uma maior robustez, já que, mesmo nos contextos em que não atingiu os melhores resultados de optimização, raramente se distanciou dos melhores resultados por várias ordens de magnitude, como aconteceu, por exemplo com o algoritmo de evolução diferencial com procura livre quando deslocámos os óptimos para a fronteira do espaço de procura.

Em termos de características específicas, podemos sublinhar algumas relacionadas não só com o comportamento e desempenho observado durante as diversas experiências, mas também as inerentes à forma como o algoritmo foi desenvolvido. Em termos de desempenho observámos que o algoritmo apresenta:

- uma velocidade rápida de convergência, especialmente em zonas monotónicas das funções, como é o caso das funções unimodais;
- capacidade para lidar convenientemente com a presença de muitos óptimos locais, mesmo que organizados de forma irregular;
- robustez à variação das funções de teste, tanto entre si, como pelo deslocamento, numa mesma função, dos óptimos para posições extremas no espaço de procura.

Quanto às características estruturais do algoritmo, decorrentes da sua construção em torno do conceito de heterogeneidade através da introdução de partículas batedoras, podemos considerar que o algoritmo demonstra:

- potencialidade para ser melhorado de forma global, pela introdução de novas partículas com mecanismos de utilidade geral, já que a introdução de novos batedores não aparenta ter custos substanciais, nem em termos computacionais nem em termos de alteração da estrutura básica do algoritmo;
- possibilidade de hibridização com outros algoritmos, já que um sub-enxame de batedores pode implementar um algoritmo de procura completamente diferente;
- flexibilidade na aplicação a novos problemas, dada a facilidade de introdução de conhecimento sobre o problema, o qual pode ser incorporado em partículas batedoras que implementam comportamentos baseados nesse conhecimento.

Voltando ao nosso problema inicial - a definição de um algoritmo de enxame capaz de um bom desempenho na optimização de máquinas de vectores de suporte - consideramos que as características observadas para o OPPB indiciam a sua adequabilidade ao problema em causa. Mais ainda, se o algoritmo na sua forma básica não conseguir ultrapassar possíveis dificuldades ainda não identificadas, parece-nos que a flexibilidade e facilidade de adaptação do algoritmo, resultante da utilização de partículas batedoras,

poderá ser fulcral na abordagem a essas dificuldades. Como nota final, sublinhe-se que a nossa procura de um algoritmo de enxame capaz de lidar com as características de um problema específico conduziu ao desenvolvimento de um algoritmo de optimização genérico, claramente competitivo com as abordagens mais recentes que esta área oferece.

Capítulo 7

Optimização de Máquinas de Vectores de Suporte

A utilidade da aplicação de algoritmos evolucionários na optimização de máquinas de vectores de suporte centra-se sobretudo em três vertentes: o treino de MVS com núcleos não positivos definidos, a optimização dos diversos parâmetros, tanto dos núcleos como da própria MVS e, finalmente, a evolução de novos núcleos especialmente adequados a problemas específicos. Neste capítulo iremos-nos concentrar na utilização de algoritmos baseados em inteligência de enxame para abordar estes diferentes aspectos, com particular ênfase no treino de máquinas de vectores de suporte com recurso a algoritmos de enxame.

Numa primeira fase adaptámos o algoritmo de optimização predador-presa com batedores ao problema específico do treino de máquinas de vector de suporte. Para tal, utilizámos os conhecimentos adquiridos durante o desenvolvimento do algoritmo para selecionar partículas batedoras promissoras, no sentido de tomar partido das características do problema para potenciar o desempenho global do algoritmo. O novo algoritmo de treino é comparado com a melhor abordagem evolucionária e dois dos métodos clássicos mais conhecidos utilizando para o efeito 10 conjuntos de dados típicos da área da aprendizagem. Apresentaremos resultados utilizando diversos tipos de função de núcleo, incluindo:

- Um núcleo PSD, a função de base radial, a qual permite observar o desempenho dos algoritmos no típico problema de optimização côncavo, contexto sobretudo útil para comparar entre si as abordagens evolucionárias.
- Núcleos não PSD, onde esperamos encontrar situações, i.e., pares problema/núcleo, em que os algoritmos evolucionários possam obter resultados superiores aos das abordagens clássicas.

- Um núcleo não PSD obtido por substituição da medida de distância, ilustrando assim a utilização desta abordagem numa situação comum em problemas práticos.

Após a avaliação do algoritmo de treino, este foi integrado num algoritmo de optimização de MVS, o qual, recebendo apenas o conjunto de dados de treino, determina um conjunto de parâmetros para cada problema, incluindo o parâmetro C e a própria função de núcleo, através da combinação linear de várias FBR. Vamos desta forma de encontro ao nosso objectivo inicial de apresentar uma abordagem integrada, baseada em algoritmos de enxame, e capaz de optimizar os diversos aspectos de uma MVS, desde a selecção do modelo, até ao treino de núcleos não PSD, passando pela escolha da própria função de núcleo. A abordagem apresentada permite basicamente automatizar a aplicação de uma máquina de vectores de suporte a um problema em que as instâncias estejam representadas em $\mathbb{R}^m$, facilitando assim a utilização deste tipo de mecanismo de aprendizagem por utilizadores inexperientes. Embora o tipo de dados de entrada e o espaço de funções de núcleo sejam limitados, são ainda discutidas algumas opções passíveis de permitir uma maior generalidade da abordagem.

7.1 Representação e Algoritmos

Sendo o treino de máquinas de vectores de suporte um problema de optimização numérica, a aplicação do algoritmo de optimização predador-presa com batedores é relativamente simples, sendo a maior dificuldade a selecção de batedores que se revelem adequados às características do problema. Em termos de representação, assumindo um problema com m exemplos, uma partícula é representada por três vectores de tamanho m, correspondendo respectivamente à posição actual da partícula, melhor posição encontrada durante a execução do algoritmo e velocidade nas m dimensões do espaço de procura. Estas estão limitadas ao intervalo $[0, C]$, onde C é o parâmetro de regularização das MVS.

$$\text{maximizar } \sum_i \alpha_i - \frac{1}{2} \sum_i \sum_j \alpha_i \alpha_j y_i y_j k(\mathbf{x}_i, \mathbf{x}_j), \tag{7.1}$$

$$\text{sujeito a } \sum_i \alpha_i y_i = 0 \tag{7.2}$$

$$\text{e } \forall i : 0 \leqslant \alpha_i \leqslant C. \tag{7.3}$$

A posição de uma partícula representa os valores α na função objectivo do problema de maximização (7.1), a qual repetimos aqui por razões de clareza, e que corresponde ao treino de uma máquina de vectores de suporte. O enxame deve assim ser utilizado

para encontrar o vector que maximiza a função, o que implica ligeiras alterações ao algoritmo base de optimização predador-presa descrito anteriormente, já que este estava implementado para problemas de minimização.

7.1.1 Um OPPB para treino de MVS

A versão do OPPB adaptada ao problema do treino de MVS é apresentada no algoritmo 7.1, incluindo as alterações necessárias para lidar com o problema de maximização e com as limitações específicas em termos de espaço de procura, i.e., com a variável correspondente a cada dimensão a variar entre 0 e C. Também a condição de terminação do algoritmo foi alterada, sendo que apenas é utilizado como limite o número máximo de iterações a realizar.

Além do algoritmo predador-presa que servirá de base ao mecanismo de treino, foi também necessário escolher as partículas batedoras, o que fizémos recorrendo ao conhecimento adquirido nas experiências descritas nos capítulos anteriores. Já estabelecemos que, em relação ao algoritmo de optimização por enxame de partículas básico, podemos esperar que o algoritmo predador-presa evite com mais facilidade a convergência prematura e consequente estagnação do algoritmo. Este aumento de variedade no enxame pode no entanto resultar numa maior lentidão na convergência para o óptimo global.

A partícula batedora de procura local, que implementámos anteriormente, permite contrabalançar o efeito descrito, nomeadamente em áreas monotónicas da função objectivo, já que verificámos que acelera consideravelmente a convergência para o óptimo nas funções de teste unimodais. Já em relação à partícula de procura por oposição, não observámos nenhum efeito generalizado que nos pareça útil para este problema em particular, de maneira que não será utilizada.

Além da partícula batedora por procura local, uma segunda via para acelerar o treino consiste na utilização de conhecimento sobre o problema embutido numa outra partícula batedora. Neste caso tivemos em conta o facto de ser conhecido que numa solução α^* para o problema de optimização muitos dos α_i terão o valor 0, correspondendo a instâncias que não são vectores de suporte. Tipicamente, apenas uma pequena percentagem das instâncias estará associada a valores de α_i diferentes de 0 e, destas, a maioria terá o valor C, isto é, o valor máximo permitido para as variáveis. Uma partícula batedora projectada para explorar este conhecimento foi já ensaiada anteriormente, tendo-se verificado que permitia melhorar o desempenho do optimizador em problemas cujas soluções partilhavam as características aqui descritas.

Algoritmo 7.1 Algoritmo predador-presa para treino de MVS.

para cada partícula i **faz**
  $\mathbf{p}_i \leftarrow \mathbf{x}_i \leftarrow \mathbf{u}(0, C)$
  se $f(\mathbf{p}_i) > f(\mathbf{p}_g)$ **então**
    $g \leftarrow i$
  fim se
  $\mathbf{v}_i \leftarrow \mathbf{u}(-C, C)$
fim para
$\mathbf{x}_p \leftarrow \mathbf{u}(0, C)$
$\mathbf{v}_p \leftarrow \mathbf{u}(-C, C)$
enquanto $t < t_{max}$ **faz**
  para cada partícula i **faz**
    para cada dimensão d **faz**
      $v_{id} \leftarrow \omega v_{id} + u(0, \phi_1)(p_{id} - x_{id}) + u(0, \phi_2)(p_{gd} - x_{id})$
      se $u(0, 1) < r \exp^{-|x_{id} - x_{pj}|}$ **então**
        $v_{id} \leftarrow v_{id} + u(-1, 1)C$
      fim se
      $v_{id} \leftarrow \text{limita}(v_{id}, -C, C)$
      $x_{id} \leftarrow x_{id} + v_{id}$
      $x_{id} \leftarrow \text{limita}(x_{id}, 0, C)$
    fim para
    se $f(\mathbf{x}_i) > f(\mathbf{p}_i)$ **então**
      $\mathbf{p}_i \leftarrow \mathbf{x}_i$
    fim se
    se $f(\mathbf{x}_i) > f(\mathbf{p}_g)$ **então**
      $g \leftarrow i$
    fim se
  fim para
  para cada dimensão d **faz**
    $v_{pd} \leftarrow \omega v_{pd} + u(0, \phi_1)(x_{gd} - x_{pd}) + u(0, \phi_2)(p_{gd} - x_{pd})$
    $x_{pd} \leftarrow x_{pd} + v_{pd}$
  fim para
fim enquanto

Neste algoritmo utilizamos como partículas batedoras a melhor partícula do enxame (batedor por procura local) e a segunda melhor partícula (batedor baseado em conhecimento). O algoritmo 7.2 descreve o comportamento de ambos os batedores, sendo que, as alterações em relação às versões utilizadas nos problemas genéricos de optimização, se centram nas alterações necessárias a um problema de maximização e na forma como é escolhida a dimensão na qual estas partículas exploram o espaço de procura. Nas

versões anteriores, a escolha da dimensão para exploração do espaço de procura era feita sequencialmente, passando por todas as dimensões. Neste algoritmo, devido ao facto de o número de iterações poder ser semelhante ou mesmo inferior ao número de dimensões, optámos por escolher essa direcção aleatoriamente. Assim, $u(m)$ devolve, com distribuição uniforme, uma dimensão das m possíveis.

A aplicação deste algoritmo a um problema de classificação descrito por um conjunto de treino que fornece os pares $\langle \mathbf{x}_i, y_i \rangle$ necessários à avaliação da função (5.1) permite assim encontrar uma partícula cuja melhor posição deverá maximizar essa mesma função. Ao longo do processo de treino da MVS, o valor da função objectivo é utilizado como medida do desempenho da partícula, à semelhança de qualquer problema de optimização. No entanto, não nos podemos esquecer de que o vector α^* é apenas o genótipo a partir do qual é construído o classificador, neste caso específico a máquina de vectores de suporte que constitui o fenótipo correspondente.

O valor da função objectivo não nos permite inferir nada sobre a qualidade do classificador final, o qual, normalmente, é avaliado pela sua capacidade em classificar correctamente novas instâncias que lhe sejam apresentadas. Uma medida possível para estimar essa capacidade é simplesmente a soma das instâncias de cada classe classificadas correctamente a dividir pela número total de instâncias do conjunto de treino. Nesta dissertação chamamos *precisão* a este valor, o qual é normalmente apresentado sobre a forma de percentagem. Da mesma forma podemos definir o *erro* de classificação como a soma das instâncias de cada classe classificadas incorrectamente a dividir pelo número total de instâncias. Estas medidas são, no entanto, tipicamente optimistas, já que, tendo o classificador sido construído com base nestas instâncias, terá com elas um desempenho melhor do que em instâncias desconhecidas.

Note-se que as máquinas de vectores de suporte são notórias por resultarem em classificadores com um desempenho elevado em novos dados, i.e., por terem uma elevada capacidade de generalização. Esta é consequência destes algoritmos, como já foi discutido, procurarem não só minimizar o risco empírico, mas também uma medida de risco estrutural. Mesmo assim, torna-se necessário utilizar um processo para estimar o desempenho de uma MVS em dados desconhecidos, já que, por exemplo, uma escolha desajustada de C, ou um conjunto de treino que represente mal o domínio das instâncias, podem resultar num classificador sobre-ajustado aos dados de treino, podendo ter como consequência uma precisão baixa em novas instâncias.

A estimativa do desempenho de algoritmos de aprendizagem é normalmente feita recorrendo à partição dos dados disponíveis em dois conjuntos, um conjunto de treino utilizado, no nosso caso, para construir um classificador, e um conjunto de teste, cujos exemplos são posteriormente utilizados para estimar a qualidade desse classificador. Neste trabalho, utilizamos como processo de validação a denominada validação cruzada *k-fold*, a qual permite, por um lado, um melhor aproveitamento dos dados disponíveis

Algoritmo 7.2 Partículas batedoras para treino de MVS.

para cada partícula i **faz**
  se $f(\mathbf{p}_i) > f(\mathbf{p}_g)$ **então**
    $k \leftarrow g$
    $g \leftarrow i$
  fim se
fim para
$d \leftarrow u(m)$
$\mathbf{x}' \leftarrow \mathbf{p}_g$
$x'_d \leftarrow x'_d + n(0, \sigma)$
se $f(\mathbf{x}') > f(\mathbf{p}_g)$ **então**
  $\mathbf{p}_g \leftarrow \mathbf{x}'$
  $s \leftarrow s + 1$
fim se
se $((t + 1) \bmod 100) = 0$ **então**
  se $s > 20$ **então**
    $\sigma \leftarrow \sigma \times 2$
  senão se $s < 20$ **então**
    $\sigma \leftarrow \sigma \div 2$
  fim se
  $s \leftarrow 0$
fim se
$d \leftarrow u(m)$
$\mathbf{x}' \leftarrow \mathbf{p}_k$
se $u(0, 1) > 0.8$ **então**
  $x'_d \leftarrow C$
senão
  $x'_d \leftarrow 0$
fim se
se $f(\mathbf{x}') > f(\mathbf{p}_k)$ **então**
  $\mathbf{p}_k \leftarrow \mathbf{x}'$
fim se
se $f(\mathbf{x}') > f(\mathbf{p}_g)$ **então**
  $g \leftarrow k$
fim se

e, por outro, diminuir a variância das estimativas de desempenho.

Na validação cruzada, o conjunto de treino é dividido em k subconjuntos de igual cardinalidade, chamados *folds*, sendo 10 o valor típico de k. Cada algoritmo é executado k vezes, com a reunião de $k-1$ subconjuntos utilizados para treinar a MVS. Depois de treinado, o classificador é aplicado ao *fold* excluído do treino. No nosso caso utilizamos a precisão ou a percentagem de erro como as medidas de desempenho do classificador, sendo que a validação cruzada utilizada devolve a média e desvio padrão obtida para as k execuções realizadas. Adicionalmente, a validação cruzada utilizada inclui um mecanismo de estratificação, o qual garante que todos os subconjuntos contêm um mesmo número de exemplos de cada classe, facto que contribui para a diminuição da variância da estimativa de desempenho, facilitando a comparação entre diferentes algoritmos.

Na sequência dos parágrafos anteriores, verifica-se que temos duas medidas de natureza diferente, ambas resultantes da execução dos algoritmos de treino das máquinas de vectores de suporte: o valor final da função objectivo e uma estimativa do desempenho, tipicamente a precisão do classificador, medida por validação cruzada. No trabalho experimental descrito nas secções seguintes, utilizamos estas medidas para comparar diferentes aspectos dos algoritmos. O valor final da função objectivo é principalmente utilizado para comparar a eficiência computacional das diferentes abordagens evolucionárias, já que permite perceber qual o número de avaliações da função objectivo necessário para atingir determinado valor dessa mesma função. A precisão é utilizada para comparar a qualidade dos classificadores gerados pelos diferentes algoritmos, tanto evolucionários como clássicos.

7.1.2 Outros algoritmos de treino

No próximo capítulo iremos comparar o desempenho do optimizador predador-presa com batedores com outros algoritmos evolucionários em diversos contextos relacionados com o treino de máquinas de vectores de suporte. Tratando-se o OPPB de um algoritmo baseado em enxames de partículas, torna-se necessário utilizar um algoritmo padrão desta área para determinar se esta nova versão apresenta realmente vantagens nas tarefas aqui discutidas. Como já referimos anteriormente, Mierswa et al [2006] implementaram uma versão do OEP básico para o treino de MVS, o qual foi testado apenas com um núcleo PSD, mais especificamente a função de base radial. Mesmo assim, o algoritmo mostrou não ser competitivo com outras abordagens evolucionárias implementadas, sendo aquelas baseadas em estratégias evolutivas. Naquela abordagem, o OEP utilizado era baseado nas equações de actualização com o peso linearmente decrescente associado à velocidade anterior [Shi and Eberhart, 1999]. Como os resultados obtidos não foram os melhores, optámos aqui por utilizar a versão canónica do algori-

tmo, proposta por Clerc [2006], com $\phi_1 = \phi_2 = 2.05$ e $\chi = 0.7298$, como termo básico de comparação. No algoritmo 7.3 apresentamos a versão do OEP básico usado, já com as adaptações decorrentes do domínio específico de procura.

Algoritmo 7.3 Algoritmo básico de OEP para treino de MVS.

para cada partícula i **faz**
  $\mathbf{p}_i \leftarrow \mathbf{x}_i \leftarrow \mathbf{u}(0, C)$
  se $f(\mathbf{x}_i) < f(\mathbf{x}_g)$ **então**
    $g \leftarrow i$
  fim se
  $\mathbf{v}_i \leftarrow \mathbf{u}(-C, C)$
fim para
enquanto $t < t_{max}$ **faz**
  para cada partícula i **faz**
    para cada dimensão d **faz**
      $v_{id} \leftarrow \chi\left(v_{id} + u(0, \phi_1)(p_{id} - x_{id}) + u(0, \phi_2)(p_{gd} - x_{id})\right)$
      $x_{id} \leftarrow x_{id} + v_{id}$
      $x_{id} \leftarrow \text{limita}(0, C)$
    fim para
    se $f(\mathbf{x}_i) < f(\mathbf{p}_i)$ **então**
      $\mathbf{p}_i \leftarrow \mathbf{x}_i$
    fim se
    se $f(\mathbf{x}_i) < f(\mathbf{p}_g)$ **então**
      $g \leftarrow i$
    fim se
  fim para
fim enquanto

O nosso objectivo principal em termos de comparação é, no entanto, testar a abordagem proposta com a melhor abordagem evolucionária encontrada na bibliografia da área. Das abordagens apresentadas em [Mierswa, 2006a], todas baseadas em estratégias evolutivas, a que apresentou consistentemente um melhor desempenho utiliza um operador de mutação gaussiano adaptativo bastante típico neste grupo de algoritmos evolucionários. É, aliás, a utilização deste operador que pode justificar a denominação do algoritmo como estratégia evolutiva, já que os outros operadores utilizados, por exemplo em termos de selecção e recombinação, poderiam perfeitamente estar incluídos num algoritmo genético. No mesmo trabalho, o autor testa duas outras variantes, mas acaba por concluir que a que aqui descrevemos é a que produz melhores classificadores, sendo esta a que utiliza em trabalho posterior [Mierswa and Morik, 2008].

Algoritmo 7.4 Estratégia evolutiva para treino de MVS.

para cada indivíduo i **faz**
  $\mathbf{x}_i \leftarrow \mathbf{u}(0, C)$
fim para
enquanto $t < t_{max}$ **faz**
  para cada indivíduo i **faz**
    $b \leftarrow u(n)$
    enquanto $s < n_{trn} - 1$ **faz**
      $k \leftarrow u(n)$
      se $f(\mathbf{x}_k) > f(\mathbf{x}_b)$ **então**
        $b \leftarrow k$
      fim se
    fim enquanto
    $\mathbf{x}'_i \leftarrow \mathbf{x}_k$
  fim para
  para cada par de indíviduos i, $i + 1$ **faz**
    para cada dimensão d **faz**
      se $u(0, 1) < 0.5$ **então**
        $x''_{i,d} \leftarrow x'_{i+1,d}$
        $x''_{i+1,d} \leftarrow x'_{i,d}$
      senão
        $x''_{i,d} \leftarrow x'_{i,d}$
        $x''_{i+1,d} \leftarrow x'_{i+1,d}$
      fim se
    fim para
  fim para
  para cada indivíduo i **faz**
    $\mathbf{x}_i \leftarrow \mathbf{x}''_i + \mathbf{n}(0, \sigma)$
  fim para
  $\sigma \leftarrow$ actualiza(σ)
fim enquanto

À imagem dos algoritmos de enxame apresentados atrás, também neste algoritmo a população inicial é gerada aleatoriamente, com cada indivíduo $\mathbf{x}_i$ correspondendo a um vector de possíveis α, os quais podem variar entre 0 e C. A selecção dos indivíduos para reprodução é feita por torneio. Para cada indivíduo $\mathbf{x}'_i$ da população de reprodução são escolhidos, aleatoriamente, 25% dos indivíduos da população anterior, dos quais o melhor vence o torneio. Opcionalmente pode também ser utilizado um mecanismo de elitismo, o qual garante que o melhor indivíduo de uma geração é garantidamente seleccionado para a seguinte.

A cada par de indivíduos $\langle \mathbf{x}'_i, \mathbf{x}'_{i+1} \rangle$ é de seguida aplicado um operador de recombinação uniforme, i.e., um operador que troca, com probabilidade 50%, cada par de genes correspondentes nos indivíduos progenitores, produzindo um novo par de descendentes $\langle \mathbf{x}''_i, \mathbf{x}''_{i+1} \rangle$. A probabilidade de recombinação é elevada, normalmente 90%. Aos indivíduos resultantes da recombinação é então aplicado o operador de mutação já referido. Todos os genes são perturbados através da adição de um valor aleatório com distribuição normal de média 0 e desvio padrão σ, o qual é inicializado com o valor $C/10$ e posteriormente actualizado durante a execução do algoritmo usando a regra do $1/5$, anteriormente descrita para a partícula batedora por procura local.

Finalmente, o mecanismo de substituição é geracional, o que não é muito comum nas estratégias evolutivas. Desta forma, todos os indivíduos da geração t são substituídos pelos seus descendentes, dando origem à geração $t + 1$. O algoritmo 7.4 resume os diversos aspectos da estratégia evolutiva utilizada.

7.1.3 Um OPPB para optimização de MVS

Após a implementação e teste do algoritmo de optimização predador-presa com batedores para treino de MVS, este foi integrado numa abordagem totalmente baseada em inteligência de enxame e capaz de optimizar os diversos aspectos das MVS usualmente tratados por abordagens evolucionárias. Embora se trate de uma abordagem que funciona sobretudo como prova de conceito, demonstrámos a sua aplicabilidade no conjunto de problemas de teste usados no nosso ambiente experimental. Posteriormente, discutiremos a sua generalização a problemas arbitrários.

Atendendo à discussão sobre aplicações de computação evolucionária na área das máquinas de vectores de suporte, os aspectos da abordagem que considerámos para optimização foram o parâmetro C, os parâmetros do núcleo e a própria função de núcleo, além do problema do próprio treino da MVS. Para este último aspecto já temos um bom algoritmo candidato, o qual permite inclusivamente lidar com núcleos que, embora úteis, não são PSD. Integrámos assim o algoritmo de treino baseado em OPPB numa abordagem de inteligência de enxame, com dois níveis, à optimização de MVS.

O objectivo da abordagem é, dado um conjunto de exemplos de treino, devolver um núcleo apropriado, o qual pode ser não PSD, os seus parâmetros e um valor de C adequado, sem intervenção do utilizador. Considerámos que o conjunto de dados, à imagem dos problemas aqui considerados, possui apenas atributos numéricos e classe binária. Central a este tipo de abordagem é a representação escolhida para o núcleo, a qual define o espaço a procurar. Neste caso optámos por utilizar uma combinação linear de núcleos FBR, a qual foi já usada com algum sucesso na bibliografia, embora limitada a núcleos PSD e, curiosamente, também baseada numa estratégia evolutiva

[Phienthrakul and Kijsirikul, 2005].

Como podemos verificar na equação (7.4), o valor devolvido pela função de núcleo é a soma pesada de n funções de base radial. No nosso caso usámos $n = 3$ e, no nível exterior da abordagem, utilizámos um algoritmo de enxame para optimizar o vector $[C, \sigma_1, \sigma_2, \sigma_3, w_1, w_2, w_3]$, onde C é o parâmetro de regularização da MVS, os parâmetros σ_i definem a forma de cada FBR individual e os pesos w_i controlam a combinação dos núcleos individuais no núcleo final. Nas nossas experiências, usámos uma codificação logarítmica (base 10) para $C \in [0.001, 1000]$ e $\sigma_i \in [0.001, 1000]$, enquanto os pesos w_i foram codificados directamente nos intervalo [-1, 1].

$$k(\mathbf{x}_i, \mathbf{x}_j) = \sum_{k=1}^{n} w_k \exp\left(\frac{-\|\mathbf{x}_i - \mathbf{x}_j\|^2}{\sigma_k}\right) \tag{7.4}$$

O facto de os pesos w_i poderem ser negativos, implica que o núcleo produzido pode ser não PSD, já que a subtração de núcleos PSD, como a função de base radial, não garante, ao contrário, por exemplo, da soma, que o núcleo resultante seja positivo semi-definido. Assim, ao contrário de abordagens anteriores, esta permite explorar um espaço alargado de núcleos, o qual pode incluir núcleos não PSD. Mas, para explorar este espaço eficientemente, é necessária a existência de algoritmos de treino das MVS resultantes que avaliem correctamente esses núcleos. Surge aqui o segundo nível de inteligência de enxame desta abordagem, no qual o OPPB descrito anteriormente é utilizado no treino dos núcleos produzidos pelo algoritmo do nível exterior.

Temos assim uma estrutura aninhada de algoritmos de enxame, com um algoritmo exterior que gere partículas que codificam possíveis núcleos e parâmetros, e um algoritmo interior que é utilizado na avaliação das soluções encontradas pelo algoritmo do nível exterior. Neste caso, o problema de optimização correspondente ao nível exterior é um problema de optimização numérica, sem características específicas, de maneira que utilizámos o OPPB básico para o abordar, com uma única partícula batedora, o batedor de procura local.

7.1.4 Ambiente experimental

Escolhemos como ambiente experimental para implementar e testar estes algoritmos o conhecido pacote de software de análise de dados RapidMiner [Mierswa et al, 2006]. Houve vários factores que levaram à escolha deste pacote específico em vez de outro software semelhante, ou mesmo de uma solução implementada de raiz:

- É um software bem conhecido em comunidades como a da aprendizagem e a da análise inteligente de dados, o que permite obter apoio com alguma facilidade.

- Já possui implementações dos algoritmos de treino baseados em estratégias evolutivas que referimos, o que, além de poupar o esforço da sua implementação, garante que usamos exactamente os algoritmos referidos na bibliografia.
- É relativamente fácil a sua expansão através da criação de novos operadores definidos pelo utilizador, os quais podem ser adicionados ao pacote principal. No nosso caso, isso permitiu-nos implementar os diversos algoritmos de treino baseados em inteligência de enxame como novos operadores.
- O código fonte é disponibilizado de forma livre, o que permite acrescentar funções impossíveis de transformar num operador. Neste trabalho, esse aspecto foi importante para a implementação de novas funções de núcleo e da sua adição a operadores já existentes.
- Também estão implementadas versões dos algoritmos de treino baseados em programação quadrática mais populares, nomeadamente o MySVM [Rüping, 2000] e o LIBSVM [Chang and Lin, 2001], os quais utilizámos também para efeitos de comparação no trabalho experimental apresentado.
- Trata-se de um pacote de aprendizagem integrado, o que implica a inclusão de muitas funcionalidades adicionais úteis a este trabalho, como sejam a gestão e avaliação das máquinas de vectores de suporte, operadores de validação, operadores de acesso e preprocessamento dos dados e muitos outros.
- Finalmente, entre os operadores disponibilizados, existe uma implementação distribuída da validação cruzada que permite acelerar substancialmente a execução de cada experiência, ao enviar o tratamento de cada *fold* para um *thread* diferente, utilizando assim mais eficientemente os actuais processadores multi-*core*.

7.2 Conjuntos de Dados

Para comparar os diferentes algoritmos, e testar o desempenho da nova abordagem proposta, seleccionámos 10 conjuntos de dados frequentemente utilizados na bibliografia da área da aprendizagem. A utilização de conjuntos de dados padrão permite-nos, além da comparação dos algoritmos propostos em problemas bem conhecidos, poder utilizar resultados obtidos da bibliografia, no sentido de possuir um termo de comparação adicional para as abordagens propostas. Por exemplo, Meyer et al [2002] apresentam uma extensa comparação empírica, utilizando MVS e 16 outros algoritmos de classificação, na qual muitos dos conjuntos de dados que seleccionámos são utilizados. Embora os resultados não possam ser directamente comparados, já que as condições experimentais são diferentes, ainda assim permitem uma validação preliminar da abordagem aqui apresentada.

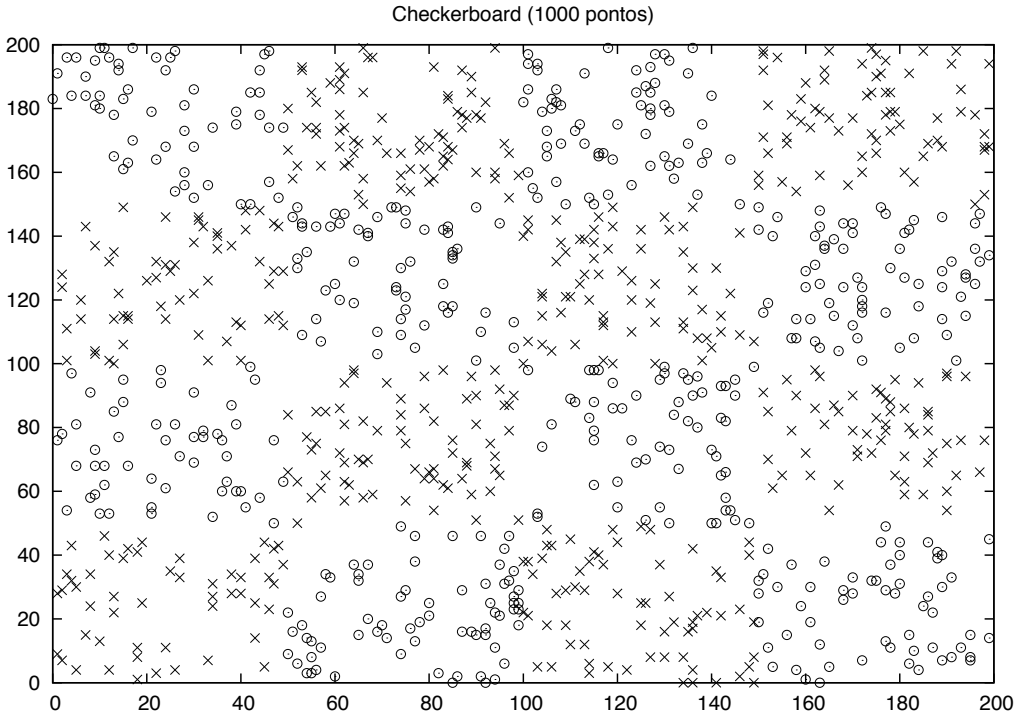


Figura 7.1: Representação gráfica do conjunto de dados *checkerboard* incluindo 1000 exemplos.

Para facilitar o trabalho experimental, e não introduzir possíveis tendências resultantes do pre-processamento dos dados, seleccionámos conjuntos de dados apenas com atributos numéricos, sem valores em falta desses atributos e correspondendo a tarefas de classificação binária, i.e., as instâncias podem apenas pertencer a duas classes diferentes. Quanto ao número de conjuntos de dados, a utilização de 10 conjuntos com características diferentes parece-nos razoável para a comparação entre os diferentes algoritmos. Note-se que o máximo utilizado até agora na bibliografia foi 8 [Mierswa and Morik, 2008] e é frequente encontrar trabalhos na área onde apenas 2 ou 3 conjuntos de dados são usados no trabalho experimental.

Os primeiros três conjuntos de dados, utilizados muito frequentemente tanto na área da aprendizagem como na investigação sobre máquinas de vetores de suporte, são conjuntos de dados gerados sinteticamente. O conjunto de dados *checkerboard* apresenta 1000 exemplos de treino organizados numa grelha de células quadradas. Como células adjacentes não podem ter instâncias da mesma classe, o aspecto da representação gráfica do conjunto de dados assemelha-se ao tabuleiro de damas ou xadrez que lhe dá o nome. Cada instância tem apenas dois atributos numéricos, de maneira que o conjunto de dados pode ser representado num plano, o que fazemos na figura 7.1.

O segundo conjunto de dados é também de utilização muito comum, consistindo em

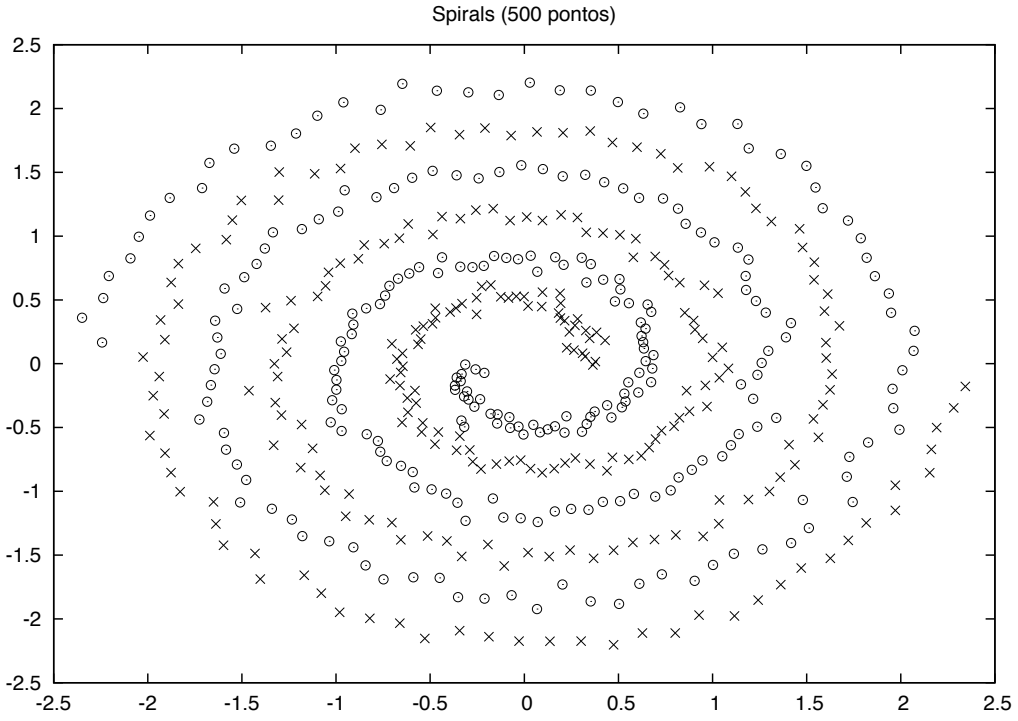


Figura 7.2: Representação gráfica do conjunto de dados *spirals* incluindo 500 exemplos.

duas espirais concêntricas de exemplos de classes diferentes. É tipicamente utilizado para ilustrar as capacidades de separação não linear das máquinas de vectores de suporte e das diferentes funções de núcleo, já que é impossível a separação com sucesso das duas espirais utilizando qualquer classificador linear. As instâncias deste conjunto de dados possuem igualmente apenas dois atributos, de maneira que também pode ser facilmente representado graficamente (ver figura 7.2).

O terceiro conjunto de dados sintético é comummente denominado *threenorm*. Neste caso cada exemplo tem 20 atributos numéricos, sendo cada ponto gerado a partir de uma distribuição normal multivariada com matriz de covariância unidade. Exemplos pertencentes à primeira classe são gerados com a mesma probabilidade a partir de duas distribuições normais, uma com média $(a, a, ..., a)$ e outra com média $(-a, -a, ..., -a)$. Os exemplos da segunda classe são obtidos utilizando uma distribuição normal com média $(a, -a, a, ..., -a)$. Em todas as distribuições temos que $a = 2/d^{0.5}$, sendo que d corresponde ao número de atributos, i.e., $d = 20$, neste caso.

Ao contrário dos dois problemas anteriores, o conjunto de dados *threenorm* resulta num problema de classificação em que existe uma substancial sobreposição nas áreas ocupadas por exemplos de classes diferentes. Este efeito pode ser observado directamente na figura 7.3, na qual podemos observar a distribuição de exemplos numa versão do

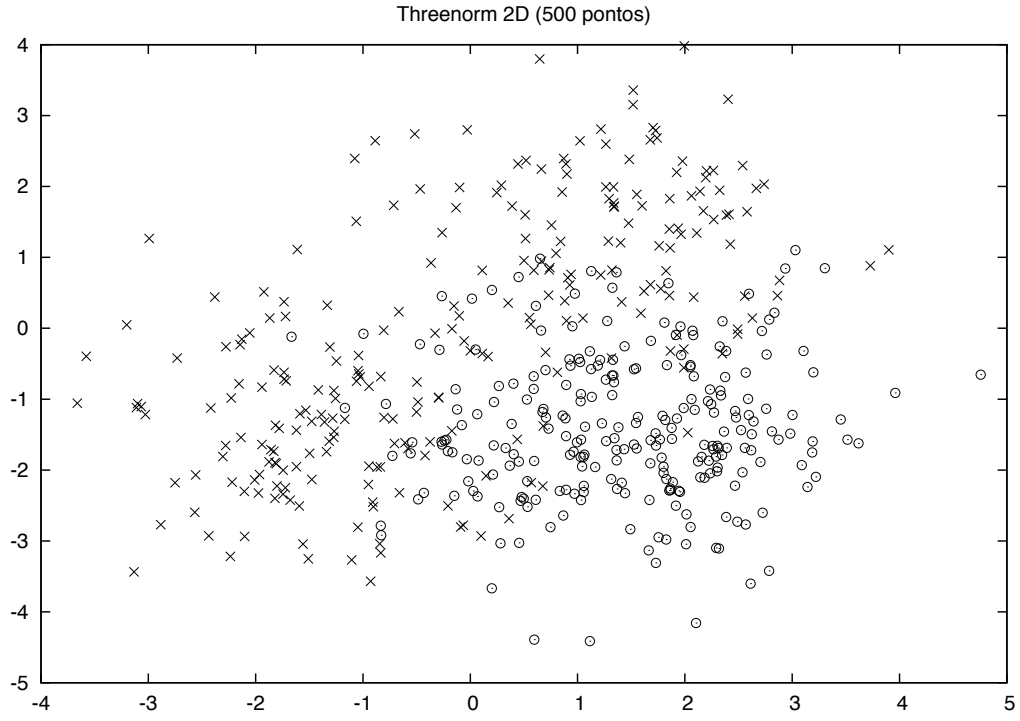


Figura 7.3: Representação gráfica do conjunto de dados *threenorm* incluindo 500 exemplos.

threenorm com apenas 2 atributos, i.e., com $d = 2$. Nesta figura, pode-se observar as três aglomerações de exemplos correspondentes às diferentes distribuições a partir de onde os pontos são gerados, bem como a sobreposição dessas mesmas aglomerações.

A maior vantagem da utilização de conjuntos de dados sintéticos centra-se na possibilidade de avaliar o desempenho dos algoritmos num ambiente controlado, no qual as dificuldades colocados por cada problema são conhecidas. É, no entanto, também importante utilizar conjuntos de dados provenientes de problemas reais, não só para determinar o desempenho dos mesmos algoritmos em situações mais próximas da sua aplicação prática, mas também para avaliar a forma como ultrapassam as dificuldades menos esperadas que estes problemas colocam. Neste trabalho utilizamos 7 conjuntos de dados resultantes de problemas reais e disponíveis em repositórios *online* de livre acesso. Procurámos escolher problemas já anteriormente utilizados no treino evolucionário de MVS, para, mais uma vez, facilitar a avaliação desta nova abordagem no contexto do trabalho já feito na área. Destes conjunto de dados, 6 foram retirados do *UCI Machine Learning Repository* [Bache and Lichman, 2013] enquanto que o restante foi retirado do *StatLib Datasets Archive* [Vlachos, 2013]. Estes conjuntos de dados são descritos brevemente nos próximos pontos.

- *Credit* - Este conjunto de dados inclui informação sobre pedidos de cartão de crédito, incluindo 690 exemplos com 14 atributos. O seu maior interesse consiste no facto de possuir atributos tanto contínuos como inteiros, sendo estes o resultado da conversão de atributos nominais. Entre os atributos com valores inteiros, existem tanto atributos com um pequeno número de valores possíveis, como atributos com um número elevado de valores possíveis, contribuindo assim para a variedade do tipo de atributos.

- *Diabetes* - Conjunto de dados relacionado com o diagnóstico de diabetes entre os índios Pima. Inclui 768 exemplos com 8 atributos contínuos e inteiros.

- *Ionosphere* - Este conjunto de dados corresponde a um problema de classificação de ecos radar reflectidos da ionosfera como resultado de emissão de pulsos de radar por 16 antenas de alta frequência colocadas no solo. O problema tem 351 instâncias com 34 atributos, tendo como aspecto mais interessante o facto dos atributos estarem agrupados aos pares, sendo que cada par corresponde a um número complexo associado ao processamento do sinal electromagnético.

- *Liver* - Contém 345 exemplos, cada um com 6 atributos, descrevendo indivíduos do sexo masculino com possíveis doenças do fígado causadas por consumo excessivo de álcool.

- *Lupus* - Este conjunto de dados contém 87 exemplos referentes a um tipo particular de Lupus, sendo utilizados apenas 3 atributos para determinar o desfecho da doença 15 anos após a sua detecção. Dos problemas reais é aquele que utiliza menos atributos para descrever uma instância.

- *Musk* - O objectivo do problema do qual resulta este conjunto de dados consiste em identificar uma nova molécula como sendo ou não um almíscar, utilizando para isto 166 atributos inteiros. Trata-se assim do problema onde um maior número de atributos é utilizado para descrever cada instância. O conjunto possui 476 instâncias de diferentes configurações de moléculas.

- *Sonar* - Este problema traduz-se num conjunto de dados com 208 instâncias descritas por 60 atributos reais, os quais descrevem um padrão de reflexão de um sinal de sonar em obstáculos que podem ser cilindros metálicos representando minas ou rochas de forma aproximadamente cilíndrica. O objectivo do problema consiste em distingir as minas das rochas através desses padrões de reflexão.

Na sua totalidade, os conjuntos de dados apresentados, são diversos em termos de origem (sintéticos e reais), em termos de número de instâncias e atributos, os quais variam substancialmente de problema para problema, e em termos do tipo de dificuldades que apresentam aos algoritmos de classificação. No caso dos problemas de origem real,

Tabela 7.1: Características dos conjuntos de dados.

Conjunto de Dados	Origem	n	m	*erro*
Checkerboard	Sintético	1000	2	48.60
Spirals	Sintético	500	2	50.00
Threenorm	Sintético	500	2	50.00
Credit	UCI MLR	690	14	44.49
Diabetes	UCI MLR	768	8	34.90
Ionosphere	UCI MLR	351	34	35.90
Liver	UCI MLR	345	6	42.03
Lupus	StatLib	87	3	40.23
Musk	UCI MLR	476	166	43.49
Sonar	UCI MLR	208	60	46.63

representam também várias vertentes de grande interesse prático para este tipo de algoritmos, e.g. na área bio-médica. Consideramos assim que constituem um conjunto de teste adequado para os algoritmos aqui discutidos.

Um resumo das características dos conjuntos de dados, incluindo a origem, o número n de instâncias e o número m de atributos, é feito na tabela 7.1. Além dessa informação, é também apresentado, na coluna *erro*, a fracção de instâncias classificadas erradamente por um classificador do tipo 0-R, i.e., um classificador que classifica todas as instâncias com a classe da maioria dos exemplos do conjunto de treino. Esta é a medida de erro base, produzida por um classificador que só usa como informação o número de exemplos de cada classe. É expectável que qualquer classificador mais sofisticado, capaz de usar a informação presente nas instâncias propriamente ditas, ultrapasse este resultado.

Capítulo 8

Resultados Experimentais

Neste capítulo apresentamos os resultado experimentais obtidos utilizando os algoritmos e conjuntos de dados descritos no capítulo anterior. Serão primeiro apresentados resultados utilizando um núcleo positivo definido, para determinar o desempenho do novo algoritmo baseado em inteligência de enxame no problema de optimização base. Pretende-se em particular avaliar se, ao contrário de abordagens anteriores, o OPPB é competitivo no problema unimodal, quando comparado com as abordagens clássicas. Posteriormente avançaremos para testes envolvendo vários núcleos não PSD, área onde há muito poucos resultados na bibliografia, e onde procuraremos determinar a utilidade dos algoritmos de evolucionários de treino de MVS e em particular do novo algoritmo proposto.

Os últimos resultados apresentados envolvem a integração do novo algoritmo de treino num algoritmo global de optimização evolucionária de máquinas de vectores de suporte, o qual, dentro de algumas limitações, permite, a partir apenas de um conjunto de dados fornecido, encontrar uma função de núcleo adequada, a qual pode ser não PSD, os seus parâmetros e o parâmetro de regularização C.

Nas experiências apresentadas de seguida discutiremos normalmente os resultados com base nas duas medidas de desempenho atrás descritas, a percentagem de erro (ou sucesso) do classificador e valor máximo encontrado para a função de optimização, obtidos através de validação cruzada. Nas experiências aqui descritas utilizaremos um número de *folds* igual a 20 (embora 10 seja o valor mais típico), para facilitar a comparação com resultados anteriores, presentes na bibliografia da área, e que usam este mesmo valor.

Em relação à percentagem de erro, verificaremos em muitas experiências que valores médios semelhantes, e desvios padrão elevados, tornam difícil avaliar se existe realmente diferença entre as abordagens utilizadas. Para facilitar essas comparações, e tirando partido dos operadores fornecidos pelo software RapidMiner, apresentamos testes de

significância estatística para esta medida. Mais concretamente, realizamos uma análise de variância ANOVA aos resultados de todos os algoritmos para determinar se é possível rejeitar a hipótese nula de todos os resultados serem estatisticamente indiferenciáveis. Consideramos um nível de confiança de 95%, i.e., rejeitamos a hipótese nula quando $p < 0.05$. No caso da análise ANOVA permitir afirmar, com a confiança definida, que há diferença significativa entre os resultados apresentados, realizamos ainda testes-t entre cada par de algoritmos, para determinar, de novo com confiança 95%, entre quais os algoritmos as diferenças de desempenho são estatisticamente significativas.

No que diz respeito aos valores máximos da função objectivo, poderemos verificar que tanto os valores médios obtidos são bastante díspares, como os desvios padrão são bastante baixos, de maneira que optámos por não realizar testes de significância estatística para esta medida. Apresentamos, no entanto, gráficos de convergência para cada uma das experiências, os quais permitem analisar o progresso desta medida para os diferentes algoritmos evolucionários. Estes gráficos permitem comparar o número de avaliações de que cada algoritmo necessita para atingir determinado valor da função objectivo, ilustrando assim as diferenças entre os algoritmos em termos de complexidade temporal.

Finalmente, nos conjuntos de experiências envolvendo as diversas funções de núcleo, optámos por utilizar $C = 1$ para todos os conjuntos de dados. Embora estejamos conscientes que esta pode não ser a escolha óptima para todos os pares conjunto de dados / função de núcleo, tomámos esta opção para, mais uma vez, facilitar a comparação com trabalho anterior, que utiliza esta definição com bons resultados, e também para simplificar a configuração experimental, evitando a optimização de um parâmetro extra. Note-se ainda que o nosso objectivo nestes primeiros conjuntos de experiências é a comparação de diversos algoritmos em circunstâncias iguais e não a obtenção dos melhores resultados possíveis. De qualquer forma, voltaremos à discussão da importância do parâmetro C no último conjunto de experiências, onde este parâmetro será optimizado em conjunto com os outros aspectos das MVS.

8.1 Resultados para a Função de Base Radial

Neste primeiro conjunto de experiências testámos as três abordagens evolucionárias descritas anteriormente, mais especificamente a melhor abordagem evolucionária anterior [Mierswa and Morik, 2008], um OEP padrão e o novo OPPB, junto com as duas abordagens clássicas mais populares, MySVM e LIBSVM, nos 10 conjuntos de dados. Utilizámos como função de núcleo a função de base radial descrita pela equação (8.1), tendo o parâmetro σ_r sido optimizado para o algoritmo MySVM em cada conjunto de dados utilizando uma procura em grelha. Os valores utilizados podem ser consultados

na tabela 8.1.

$$k(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(\frac{-\|\mathbf{x}_i - \mathbf{x}_j\|^2}{\sigma_r}\right) \tag{8.1}$$

Tabela 8.1: Valores de σ_r utilizados para a função de base radial em cada conjunto de dados.

Conjunto de Dados	σ_r
Checkerboard	100
Spirals	100
Threenorm	1
Credit	0.001
Diabetes	0.1
Ionosphere	0.1
Liver	1
Lupus	0.1
Musk	0.1
Sonar	0.1

Uma vez que a função de núcleo utilizada é positiva definida, o problema é unimodal e as principais dificuldades para as abordagens evolucionárias deverão ser a elevada dimensionalidade da função de optimização e o facto de um número elevado dos α_i deverem ser 0 ou C, colocando assim o óptimo na fronteira do espaço de procura em muitas dimensões. Os algoritmos evolucionários foram executados durante 500 iterações, utilizando 20 indivíduos/partículas, com excepção do OPPB, no qual só foram usadas 18 partículas para compensar as avaliações extra das partículas batedoras. Os resultados obtidos para cada par algoritmo/conjunto de dados, em termos de erro de classificação médio e respectivo desvio padrão, bem como o valor p do teste ANOVA, são apresentados na tabela 8.2.

Tabela 8.2: Erro dos classificadores utilizando a função de base radial.

Conjunto de dados	MySVM	LIBSVM	EE	OEP	OPPB	p
Checkerboard	5.6 (3.0)	5.6 (3.8)	5.9 (3.0)	6.9 (3.3)	5.9 (3.7)	0.737
Spirals	0.2 (0.9)	0.2 (0.9)	0.6 (1.4)	3.8 (3.5)	0.4 (1.2)	**0.000**
Threenorm	14.6 (8.1)	13.6 (6.6)	13.2 (8.2)	12.8 (9.1)	14.4 (7.0)	0.943
Credit	16.8 (8.7)	14.5 (5.4)	13.8 (4.1)	13.3 (6.4)	14.4 (5.7)	0.465
Diabetes	22.8 (5.2)	23.2 (5.8)	23.3 (6.1)	28.4 (7.6)	23.4 (7.8)	**0.045**
Ionosphere	6.3 (5.1)	6.0 (3.8)	8.3 (5.8)	22.6 (9.7)	6.8 (6.1)	**0.000**
Liver	29.3 (8.4)	30.2 (5.8)	28.1 (10.9)	30.1 (10.4)	27.8 (8.7)	0.879
Lupus	25.3 (20.4)	26.2 (16.3)	26.0 (16.3)	22.0 (17.6)	24.0 (16.2)	0.939
Musk	7.8 (4.9)	7.3 (4.7)	8.8 (5.3)	11.8 (5.6)	8.6 (6.1)	0.085
Sonar	14.4 (9.1)	14.5 (7.9)	12.3 (9.7)	15.2 (12.4)	14.3 (11.9)	0.729

Uma análise dos valores de erro apresentados na tabela 8.2 permite chegar a várias conclusões:

- Apenas para três conjuntos de dados (*Spirals*, *Diabetes* e *Ionosphere*) encontramos diferenças estatisticamente significativas entre o desempenho dos diferentes algoritmos. Olhando para os resultados dos testes-t para cada um desses conjuntos de dados, apresentados nas tabelas 8.3, 8.4 e 8.5, podemos verificar que em todos os casos é o algoritmo OEP básico que tem um desempenho inferior ao de vários dos outros algoritmos. Estes, por sua vez, têm todos desempenhos

estatisticamente indiferenciáveis entre si.

- Estes resultados, no que diz respeito ao OEP padrão, são uma confirmação dos resultados obtidos por Mierswa [2006a], que sublinhou as debilidades dos optimizadores de enxame básicos quando em competição tanto com as abordagens clássicas, como com os melhores algoritmos evolucionários, no treino de máquinas de vectores de suporte, mesmo quando são usados núcleos PSD. Voltaremos às razões deste desempenho menos bom quando analisarmos o comportamento dos algoritmos em termos do problema de optimização.

- Tanto em termos de precisão (percentagem de erro), como de robustez (desvio padrão), não há diferença significativa entre as abordagens clássicas e as abordagens evolucionárias, o que demonstra que tanto a abordagem baseada em estratégias evolutivas, como o OPPB, são perfeitamente capazes de lidar com a versão unimodal do problema de treino das MVS. Estes resultados também sugerem que, para o mesmo problema e utilizando os mesmos parâmetros, todas as abordagens encontram soluções com a mesma qualidade.

- A versão do optimizador predador-presa com batedores aqui proposta é o primeiro algoritmo baseado em inteligência de enxame claramente competitivo com outras abordagens evolucionárias neste problema, pelo menos em termos da qualidade do optimizador final obtido.

Os valores de precisão apresentados são relevantes para a avaliação da qualidade dos classificadores finais produzidos por cada algoritmo. No entanto, inerente ao treino de cada classificador, está a resolução de um problema de optimização. Os resultados finais, em termos dos valores da função objectivo, são importantes para melhor podermos comparar o comportamento dos algoritmos evolucionários neste problema. Na tabela 8.6 apresentamos os valores médios e respectivos desvios padrão de $f(\alpha^*)$ para cada par algoritmo evolucionário / conjunto de dados, i.e., o valor da função objectivo para o melhor vector α^* encontrado por cada algoritmo ao longo da sua execução. Lembramos que os problemas de optimização resolvidos são problemas de maximização e que, por isso, valores mais altos correspondem a um melhor desempenho do algoritmo. Para facilitar a análise, e melhor compreender como os resultados finais foram obtidos, apresentamos também, nas figuras 8.1 e 8.2 os gráficos de convergência dos três algoritmos para os 10 problemas de optimização.

Tabela 8.3: Valores de p obtidos nos testes-t realizados para cada par de algoritmos, usando a função de base radial no conjunto de dados *Spirals*.

	LIBSVM	EE	OEP	OPPB
MySVM	1.000	0.423	**0.000**	0.556
LIBSVM	-	0.423	**0.000**	0.556
EE	-	-	**0.000**	0.639
OEP	-	-	-	**0.000**

Tabela 8.4: Valores de p obtidos nos testes-t realizados para cada par de algoritmos, usando a função de base radial no conjunto de dados *Diabetes*.

	LIBSVM	EE	OEP	OPPB
MySVM	0.824	0.776	**0.011**	0.775
LIBSVM	-	0.948	**0.023**	0.923
EE	-	-	**0.030**	0.969
OEP	-	-	-	0.056

Tabela 8.5: Valores de p obtidos nos testes-t realizados para cada par de algoritmos, usando a função de base radial no conjunto de dados *Ionosphere*.

	LIBSVM	EE	OEP	OPPB
MySVM	0.839	0.363	**0.000**	0.758
LIBSVM	-	0.194	**0.000**	0.604
EE	-	-	**0.000**	0.461
OEP	-	-	-	**0.000**

Tabela 8.6: Valor máximo $f(\alpha^*)$ obtido para a função objectivo por cada um dos algoritmos evolucionários, utilizando a função de base radial.

Conjunto de dados	EE	OEP	OPPB
Checkerboard	116.0 (4.7)	-46.7 (20.3)	159.1 (2.7)
Spirals	102.6 (0.9)	80.0 (3.9)	102.4 (0.9)
Threenorm	134.5 (1.9)	94.5 (4.3)	139.8 (1.6)
Credit	330.3 (3.1)	275.5 (6.2)	369.5 (3.0)
Diabetes	252.0 (10.1)	-192.2 (92.9)	369.0 (5.9)
Ionosphere	58.8 (3.7)	-182.1(75.5)	76.0 (1.4)
Liver	197.0 (2.9)	147.5 (6.3)	228.3 (2.0)
Lupus	47.6 (1.2)	40.7 (3.4)	51.4 (1.1)
Musk	127.4 (1.5)	90.1 (4.4)	129.9 (1.8)
Sonar	67.1 (1.2)	49.7 (2.5)	71.1 (1.6)

Os resultados em termos do problema de optimização são mais fáceis de analisar, já que há maiores diferenças entre os resultados de cada algoritmo e os desvios padrão são também bastante menores. Também aqui são várias as conclusões que podemos retirar:

- Tendo em conta apenas os valores finais da tabela 8.6, verifica-se que o OEP obtém geralmente resultados, em termos da função objectivo, muito piores do que os dois outros algoritmos. Este facto verifica-se tanto em termos dos valores médios, como também dos desvios padrão, o que indica, além de um desempenho pobre, também alguma falta de robustez. Curiosamente, os resultados são inferiores em praticamente todos os problemas e não apenas naqueles em que já verificámos que este algoritmo produziu classificadores mais pobres do que os restantes. Este resultado está ligado ao facto de nem sempre ser necessário uma MVS óptima (com o hiperplano mais discriminador) para produzir bons resultados.

- Olhando para os gráficos de convergência, torna-se notória a razão deste comportamento. O algoritmo, de início, converge rapidamente, com uma subida abrupta do valor da função objectivo, mas logo estagna, não se verificando melhoria adicional no restante das iterações. Esta fase de estagnação, que em todos os problemas

se segue logo às primeiras dezenas de iterações, é consistente com a convergência prematura do algoritmo e a sua incapacidade em fazer o enxame escapar da vizinhança de uma solução sub-óptima para uma nova fase de exploração.

- As dificuldades encontradas pelo OEP padrão estão de acordo com os pontos fracos do algoritmo que discutimos aquando da sua apresentação, especialmente a dificuldade em gerir o equilíbrio entre procura local e global e a inexistência de um mecanismo do tipo mutação que permita reintroduzir variedade no enxame, pontos fracos esses que procurámos atenuar no OPPB.

- Essas alterações parecem ter produzido os objectivos desejados já que, olhando ainda apenas para os resultados finais, o OPPB é o algoritmo que melhores valores finais da função objectivo apresenta para todos os conjuntos de dados, com exceção do conjunto *Spirals* onde são iguais. Também ao nível dos desvios padrão, os valores são iguais ou menores do que os obtidos pela EE, indicando um robustez superior.

- Mais significativa ainda é a análise dos gráficos de convergência. Podemos neles verificar que os dois algoritmos se comportam de forma bastante diferente. O OPPB tem uma convergência inicial acentuada, tal como o OEP, mas evita os pontos fracos deste, mantendo o incremento da qualidade da solução durante muitas mais iterações. Já a EE tem uma convergência inicial mais lenta, mantendo, no entanto, níveis mais elevados de melhoria durante mais tempo. Em alguns problemas (e.g. *Credit*, *Liver*, *Lupus*, *Sonar*), no entanto, parece convergir prematuramente para soluções sub-óptimas (ou pelo menos inferiores às encontradas pelo OPPB).

- O ritmo mais lento de convergência da EE implica também que, em muitos dos problemas, o OPPB é mais eficiente do ponto de vista computacional do que a EE utilizada. Analisando os gráficos apresentados, mesmo de forma qualitativa, podemos concluir que o algoritmo de enxame é capaz de encontrar soluções com a mesma qualidade do que as obtidas no fim da execução da EE, utilizando geralmente menos de metade das avaliações da função objectivo (e, em vários casos, muito menos).

Tomadas no seu conjunto, estas conclusões aparentam suportar as decisões tomadas no projecto desta versão do OPPB. As partículas batedoras aparentam contribuir para uma rápida convergência do algoritmo, enquanto que o mecanismo predador-presa contribui para o controlo do equilíbrio procura global / procura local e para a manutenção da diversidade, evitando a convergência prematura. O algoritmo resultante parece poder ser comparado favoravelmente com o melhor algoritmo evolucionário, pelo menos neste caso em que os problemas de optimização são unimodais.

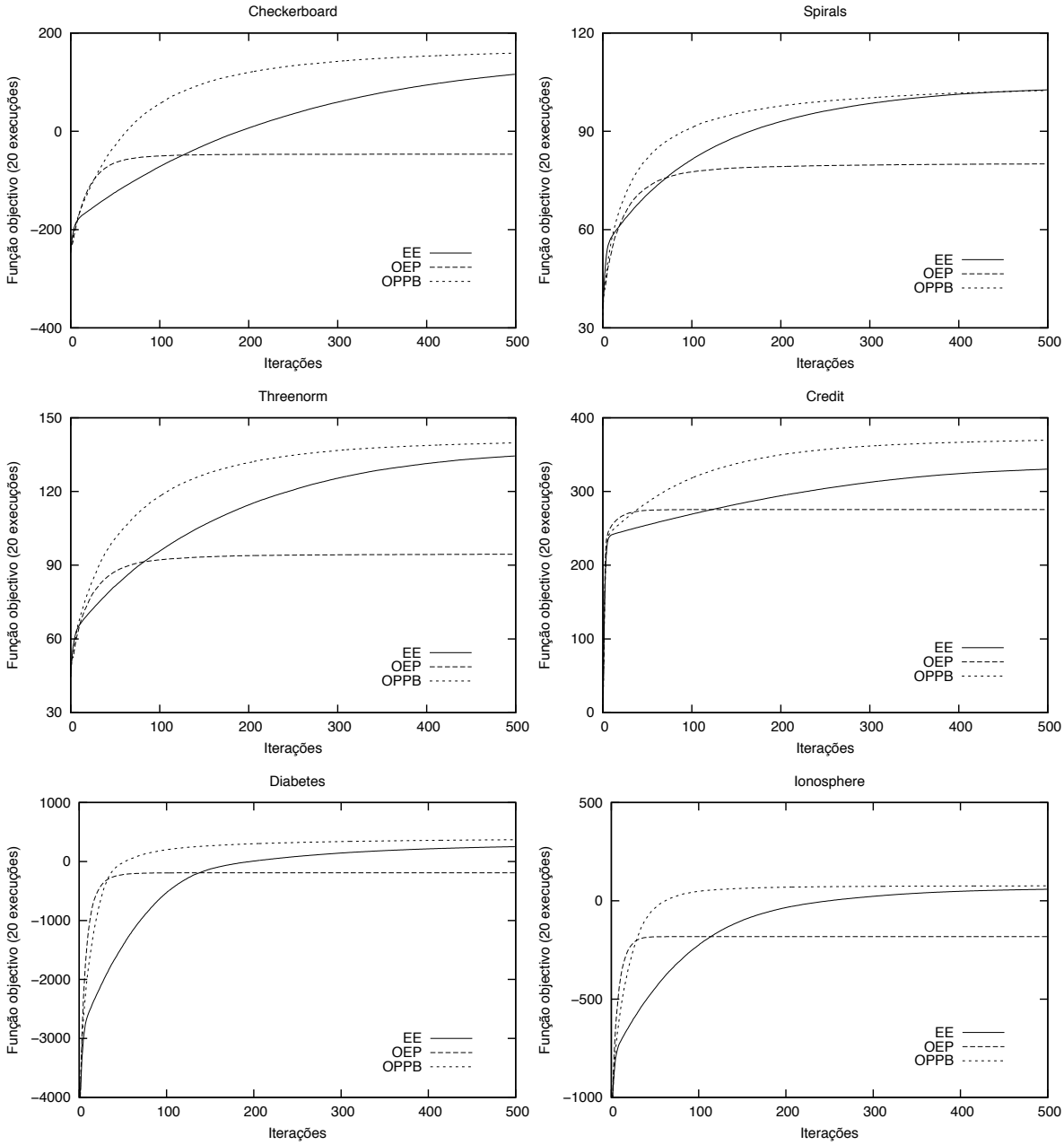


Figura 8.1: Gráficos de convergência da função objectivo para os conjuntos de dados *Checkerboard*, *Spirals*, *Threenorm*, *Credit*, *Diabetes* e *Ionosphere*, utilizando a função de base radial como função de núcleo.

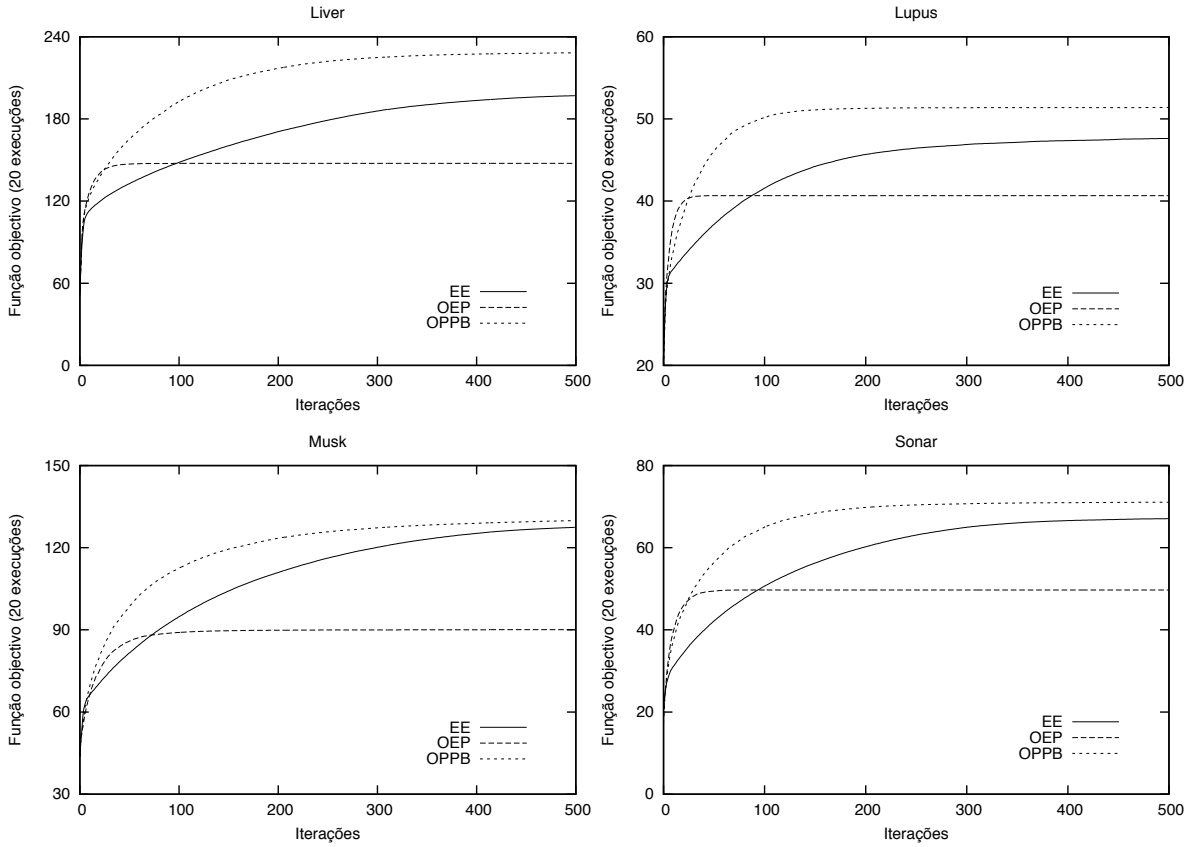


Figura 8.2: Gráficos de convergência da função objectivo para os conjuntos de dados *Liver*, *Lupus*, *Musk* e *Sonar*, utilizando a função de base radial como função de núcleo.

Nas próximas secções testaremos os algoritmos com funções de núcleo não PSD, podendo assim verificar se o OPPB mantém o mesmo nível de desempenho em problemas onde poderá encontrar funções com mais de um óptimo. Serão também estes problemas que permitirão determinar a relevância prática destes algoritmos de treino, já que não há obviamente nenhuma vantagem particular em utilizar algoritmos evolucionários para treinar MVS com núcleos PSD, circunstância em que os métodos clássicos são mais eficientes.

8.2 Resultados para a Função de Núcleo de Epanechnikov

Neste segundo conjunto de experiências, comparamos as duas melhores abordagens evolucionárias identificadas na secção anterior com uma das abordagens clássicas, utilizando os mesmos conjuntos de dados e substituindo a função de base radial pela função de núcleo de Epanechnikov. Deixámos de incluir o OEP padrão no ambiente

experimental, já que de acordo com os resultados anteriores, este método tem dificuldades em optimizar adequadamente mesmo o problema unimodal, logo não há razões para assumir que tenha um desempenho superior às outras abordagens quando o núcleo é não PSD. Adicionalmente, a eliminação de um algoritmo das experiências torna mais rápida a execução das mesmas. Também por esta última razão passamos a utilizar apenas um dos algoritmos clássicos, nestes caso o MySVM. Como ficou claro das experiências anteriores com diversos conjuntos de dados, ambas as abordagens têm desempenho semelhante num mesmo problema, quando os mesmos parâmetros são usados, i.e., quando o problema de optimização é igual. A escolha do MySVM em detrimento do LIBSVM tem apenas a ver com questões práticas relacionadas com a utilização de diferentes núcleos no RapidMiner.

Tabela 8.7: Valores de σ_e e d utilizados para a função de núcleo de Epanechnikov em cada conjunto de dados.

Conjunto de dados	σ_e	d
Checkerboard	0.92	6.54
Spirals	0.12	3.84
Threenorm	61.60	9.38
Credit	564.62	0.65
Diabetes	220.51	4.87
Ionosphere	2.44	7.48
Liver	61.59	6.90
Lupus	241.63	7.42
Musk	63.12	6.93
Sonar	61.63	6.90

A função de núcleo de Epanechnikov foi escolhida como primeiro núcleo não PSD a ser utilizado neste trabalho por ter sido o núcleo investigado no trabalho anterior relacionado com o treino evolucionário de MVS com núcleos não PSD [Mierswa and Morik, 2008]. Esta função de núcleo calcula a similaridade entre os dois vectores reais $\mathbf{x}_i$ e $\mathbf{x}_j$ a partir da distância entre os dois, conforme especificado pela equação (8.2). Este núcleo tem dois parâmetros diferentes, σ_e e d, cujos valores utilizados para cada

conjunto de dados estão listados na tabela 8.7.

$$k(\mathbf{x}_i, \mathbf{x}_j) = \begin{cases} \left(1 - \frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{\sigma_e}\right)^d, & \text{se } \frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{\sigma_e} \leq 1; \\ 0, & \text{se } \frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{\sigma_e} > 1. \end{cases} \tag{8.2}$$

Tabela 8.8: Erro dos classificadores utilizando a função de núcleo de Epanechnikov.

Conjunto de dados	MySVM	EE	OPPB	p
Checkerboard	6.5 (4.6)	8.5 (4.3)	7.8 (4.4)	0.352
Spirals	7.2 (3.9)	13.6 (7.6)	7.8 (5.3)	0.280
Threenorm	14.0 (7.5)	15.2 (8.2)	14.0 (5.4)	0.832
Credit	14.2 (6.4)	13.3 (6.8)	13.8 (7.5)	0.925
Diabetes	24.2 (3.5)	26.9 (7.6)	25.3 (5.0)	0.298
Ionosphere	26.3 (10.3)	25.9 (10.2)	**15.7 (5.9)**	**0.000**
Liver	42.1 (2.8)	36.8 (10.8)	**35.4 (10.6)**	**0.042**
Lupus	28.2 (14.6)	22.5 (16.1)	**19.5 (12.3)**	0.156
Musk	7.8 (3.8)	11.3 (5.2)	8.7 (6.5)	0.087
Sonar	12.4 (11.4)	12.9 (10.2)	12.5 (11.0)	0.989

Nas experiências realizadas com núcleos não PSD utilizámos uma estratégia diferente para a escolha dos parâmetros. Em vez da procura em grelha que encontraria os parâmetros ideais para um dos métodos, utilizámos o operador de optimização evolucionária de parâmetros do RapidMiner. O operador foi executado de forma breve para cada problema (10 gerações e 10 indivíduos), utilizando a estratégia evolutiva como algoritmo de treino. O objectivo foi encontrar parâmetros razoáveis para cada problema, que não beneficiassem particularmente a estratégia de procura de um dos algoritmos. Com este método, a haver um algoritmo em vantagem seria a EE usada na optimização dos parâmetros, mas procurámos minorar este efeito através do número limitado de avaliações.

Foi possível observar nos gráficos de convergência das experiências anteriores que os

algoritmos evolucionários se aproximavam da convergência muito antes das 500 iterações usadas como limite. De forma a acelerar a execução das experiências, e após algumas experiências preliminares, decidimos utilizar um limite de 150 iterações para os problemas sintéticos e de 100 iterações para os restantes. Na tabela 8.8 apresentamos o erro de classificação médio, respectivo desvio padrão e valor de p para os três algoritmos mencionados, nos 10 conjuntos de teste. Os resultados dos testes-t entre algoritmos são apresentados nas tabelas 8.9 e 8.10. A tabela 8.11 apresenta, apenas para as abordagens evolucionárias, o melhor valor médio encontrado para a função objetivo, $f(\alpha^*)$.

Tabela 8.9: Valores de p obtidos nos testes-t realizados para cada par de algoritmos, usando o núcleo de Epanechnikov no conjunto de dados *Ionosphere*.

	EE	OPPB
MySVM	0.905	**0.000**
EE	-	**0.000**

Tabela 8.10: Valores de p obtidos nos testes-t realizados para cada par de algoritmos, usando o núcleo de Epanechnikov no conjunto de dados *Liver*.

	EE	OPPB
MySVM	**0.047**	**0.000**
EE	-	0.591

Este segundo conjunto de resultados permite-nos também retirar várias conclusões:

- Em primeiro lugar, todos os algoritmos foram capazes de aprender com o núcleo não-PSD, apresentando, na generalidade dos pares conjunto de dados / algoritmo, percentagens de erro inferiores aos erros base listados na tabela 7.1. A única excepção foi o par MySVM / *Liver* onde o algoritmo clássico não consegue superar o erro base, embora ambos os algoritmos evolucionários o consigam fazer. Inclusivamente, para o conjunto de dados *Lupus*, é obtido um melhor resultado utilizando o núcleo de Epanechnikov (e o algoritmo OPPB) do que utilizando a função de base radial com qualquer um dos algoritmos testados.

- Em segundo lugar, podemos observar que para três conjuntos de dados há diferenças substanciais entre o desempenho dos algoritmos, sendo que em dois deles (*Ionosphere*, *Liver*) essa diferença é estatisticamente significativa. No terceiro conjunto (*Liver*) a diferença é grande, mas um desvio padrão elevado não permite a relevância estatística. Tendo em atenção as tabelas com os resultados dos testes-t entre algoritmos (8.9, 8.10), é visível que o desempenho do OPPB é superior ao do algoritmo clássico em ambos os casos.

- Uma vez que, a partir das experiências anteriores, sabemos que os algoritmos são capazes de obter resultados idênticos quando se utilizam os mesmos parâmetros nos problemas em que a função objectivo é côncava, as diferenças de desempenho aparentemente correspondem a situações em que, tendo o núcleo utilizado produzido um problema multimodal, a abordagem baseada em programação quadrática não consegue convergir para o óptimo global de forma tão consistente como o algoritmo de enxame.

- Finalmente, entre as abordagens evolucionárias, existem também algumas diferenças de desempenho, em termos de precisão dos classificadores, com vantagem do OPPB em várias situações. Aqui, a explicação, discutida de seguida, está relacionada não tanto com a adequação do algoritmo aos problemas, mas mais com a limitação dos recursos computacionais disponíveis, i.e., com a diminuição do número de iterações realizadas.

Na tabela 8.11 podemos verificar que, em termos do valor máximo $f(\alpha^*)$ obtido para a função objectivo, o OPPB obteve sistematicamente resultados superiores aos da estratégia evolutiva. Note-se ainda que, em vários dos conjuntos de dados (sobretudo *Checkerboard*, *Threenorm* e *Diabetes*), os resultados obtidos pela EE estão associados a desvios padrão muito mais elevados do que os resultantes da utilização do algoritmo de enxame. Esta falta de robustez pode estar associada a uma maior dificuldade da EE em escapar a óptimos locais.

Tendo em atenção os gráficos de convergência apresentados nas figuras 8.3 e 8.4, fica claro que os resultados nos problemas de optimização estão relacionados com a eficiência computacional de cada um dos algoritmos evolucionários. Com a imposição de limites mais estritos ao número de avaliações disponíveis, torna-se mais difícil à EE atingir valores perto do óptimo que resultem em classificadores com desempenho comparável aos produzidos pelo OPPB. Em todos os gráficos é observável que o algoritmo de optimização predador-presa com batedores, à imagem das experiências anteriores, necessita de significativamente menos avaliações da função objectivo para alcançar valores dessa função semelhantes aos melhores encontrados pela estratégia evolutiva. Note-se que este facto é igualmente verdade em muitos dos problemas em que os classificadores produzidos por cada algoritmo tem desempenho semelhante, o que sugere

que poderíamos ainda ter ganhos substancias do ponto de vista computacional para o OPPB, reduzindo o limite de iterações, sem perder qualidade nas MVS produzidas.

Tabela 8.11: Valor máximo $f(\alpha^*)$ obtido para a função objectivo por cada um dos algoritmos evolucionários, utilizando o núcleo de Epanechnikov.

Conjunto de dados	EE	OPPB
Checkerboard	-303.6 (32.6)	51.5 (9.2)
Spirals	163.5 (3.0)	188.4 (2.8)
Threenorm	24.5 (17.0)	133.7 (4.0)
Credit	253.4 (5.7)	300.8 (4.3)
Diabetes	-114.4 (158.8)	296.1 (8.1)
Ionosphere	81.1 (3.7)	100.1 (1.9)
Liver	180.1 (6.0)	228.9 (5.0)
Lupus	49.0 (2.0)	58.5 (0.8)
Musk	101.8 (4.2)	117.4 (3.2)
Sonar	50.5 (1.9)	61.8 (1.56)

O aspecto mais interessante a retirar destas conclusões tem a ver com o facto de, ao utilizarmos um núcleo não PSD, como a função de núcleo de Epanechnikov, ser possível encontrar problemas em que os algoritmos evolucionários têm um melhor desempenho do que as abordagens clássicas, i.e., produzem classificadores mais precisos do que os algoritmos baseados em programação quadrática. Este resultado confirma o nosso pressuposto inicial de que as abordagens evolucionárias podem ser ferramentas úteis no treino de MVS e reforça resultados mais limitados obtidos por Mierswa and Morik [2008]. Nas próximas secções, investigaremos se estes resultados são extensíveis a outros núcleos não PSD.

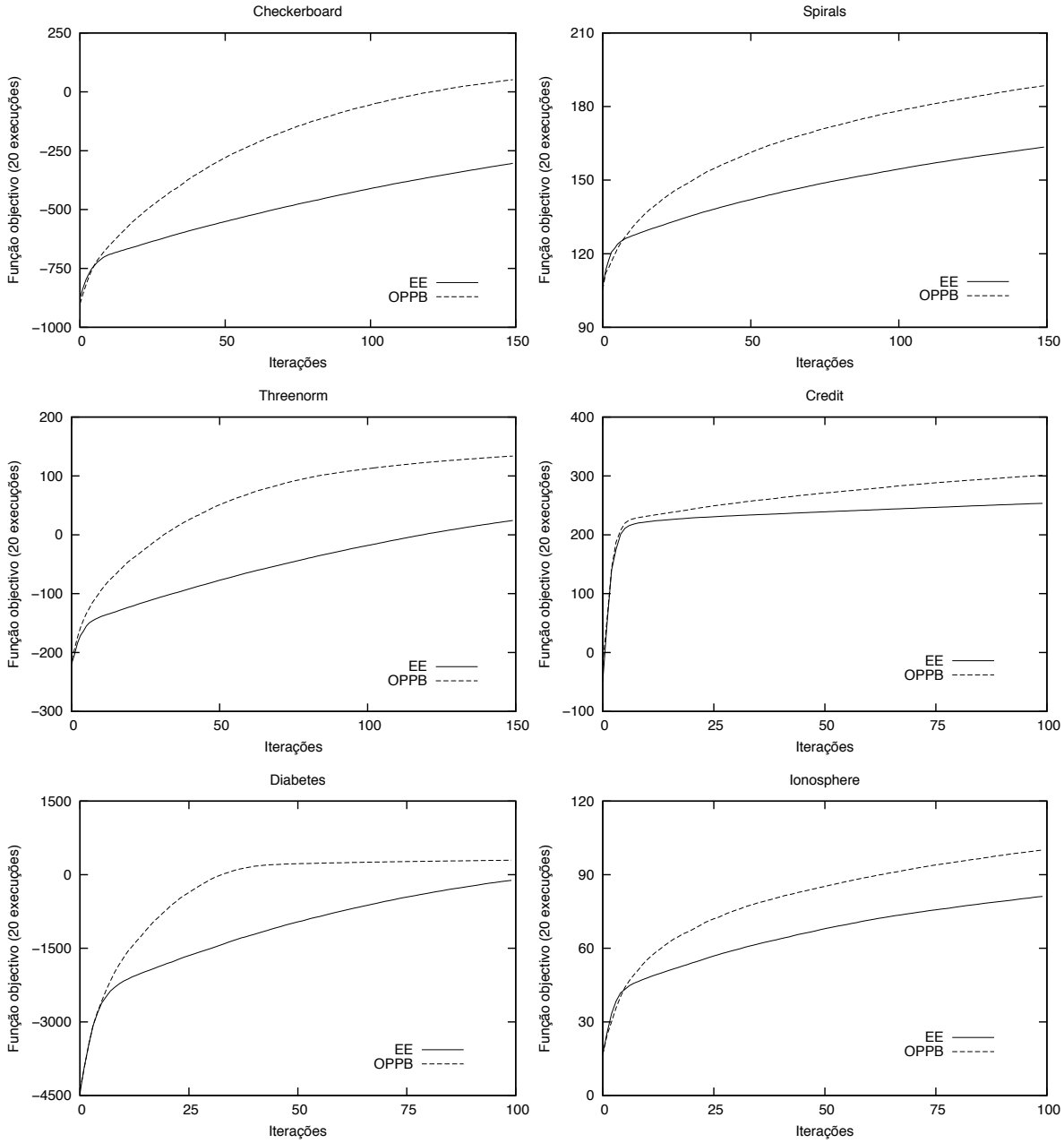


Figura 8.3: Gráficos de convergência da função objectivo para os conjuntos de dados *Checkerboard*, *Spirals*, *Threenorm*, *Credit*, *Diabetes* e *Ionosphere*, utilizando a função de núcleo de Epanechnikov.

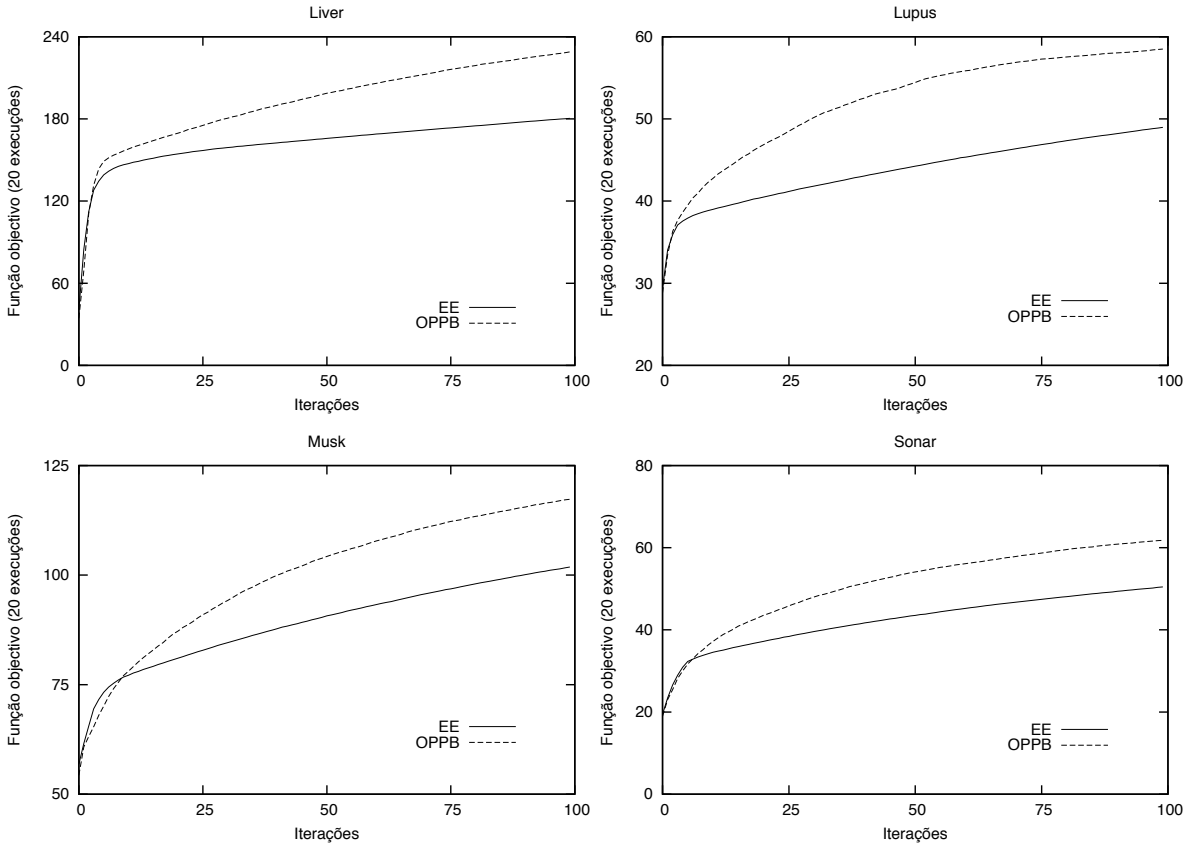


Figura 8.4: Gráficos de convergência da função objectivo para os conjuntos de dados *Liver*, *Lupus*, *Musk* e *Sonar*, utilizando a função de núcleo de Epanechnikov.

8.3 Resultados para a Função de Núcleo Sigmóide

O núcleo sigmóide, apresentado na equação (8.3), teve durante algum tempo uma utilização bastante disseminada nas máquinas de vectores de suporte, devido à sua popularidade na área das redes neuronais. Apesar de ser possível, com esta função de núcleo, obter resultados semelhantes aos que a FBR permite, esta função tem vindo a perder popularidade devido ao facto de poder ser não PSD para um subconjunto de valores dos seus parâmetros [Lin and Lin, 2003]. Neste conjunto de experiências, os valores dos parâmetros a e b foram obtidos utilizando a metodologia descrita na secção anterior e são listados, para cada conjunto de dados, na tabela 8.12. As restantes definições experimentais são as já descritas na secção anterior, exceptuando o limite de iterações, o qual foi fixo em 50 para os problemas sintéticos e 30 para os restantes.

$$k(\mathbf{x}_i, \mathbf{x}_j) = \tanh\left(a\langle \mathbf{x}_i, \mathbf{x}_j \rangle - b\right) \tag{8.3}$$

Tabela 8.12: Valores de a e b utilizados para a função de núcleo sigmóide em cada conjunto de dados.

Conjunto de dados	a	b
Checkerboard	1.258	1.452
Spirals	1.119	-1.737
Threenorm	0.110	1.746
Credit	0.378	-1.894
Diabetes	0.616	-0.065
Ionosphere	0.132	-0.318
Liver	0.116	0.770
Lupus	0.120	0.786
Musk	0.006	0.197
Sonar	0.025	0.752

Os resultados obtidos, em termos de precisão dos classificadores (tabela 8.13), reforçam, em geral, as conclusões retiradas a partir das experiências descritas na secção anterior. A maior diferença tem a ver com o facto de, em três problemas (*Checkerboard*, *Spirals* e *Liver*), nenhum algoritmo ter conseguido produzir MVS capazes de ultrapassar o erro base, i.e., não foram capazes de aprender. Tendo em atenção outros trabalhos que utilizam esta função de núcleo, é provável que se possa atribuir este resultado ao facto de não termos optimizado o valor de C [Lin and Lin, 2003]. Esta constatação não impede, no entanto, a comparação dos resultados nos restantes problemas.

Nesses podemos verificar que existem diferenças substanciais em cinco dos problemas, sempre com os classificadores de maior precisão a serem produzidos pelo algoritmo baseado em inteligência de enxame, acompanhado, em alguns deles, pelo outro algoritmo evolucionário. Em três dos cinco problemas (*Credit*, *Diabetes* e *Ionosphere*), as diferenças são estatisticamente significativas, enquanto nos restantes os valores elevados de variação, traduzidos nos desvios padrão elevados, não permitem, mais uma vez, que essa significância seja atingida. Recorrendo aos valores dos testes-t, apresentados nas tabelas 8.14, 8.15 e 8.16, verifica-se que também aqui o algoritmo de enxame é superior, com significância estatística, ao MySVM nos três problemas. A EE é superior à abordagem clássica nos dois primeiros problemas, mas não consegue o mesmo resultado

no conjunto de dados *Ionosphere*.

Também com a utilização desta função de núcleo se notam algumas diferenças em termos de precisão, quando comparamos as duas abordagens evolucionárias, embora essas diferenças só sejam significativas num único problema (*Ionosphere*). Veremos de seguida que, ao nível do problema de optimização, as diferenças são mais substanciais.

Tabela 8.13: Erro dos classificadores utilizando a função de núcleo sigmóide.

Conjunto de dados	MySVM	EE	OPPB	p
Checkerboard	48.3 (0.8)	48.5 (6.2)	48.0 (7.1)	0.976
Spirals	51.6 (10.7)	50.0 (8.0)	49.8 (10.4)	0.819
Threenorm	16.6 (9.1)	16.8 (8.7)	17.0 (6.0)	0.988
Credit	22.2 (7.4)	14.5 (5.5)	**14.1 (5.9)**	**0.000**
Diabetes	38.7 (5.3)	30.5 (7.4)	**27.2 (7.7)**	**0.000**
Ionosphere	19.9 (6.8)	25.9 (8.5)	**12.8 (8.6)**	**0.000**
Liver	42.1 (2.8)	41.8 (5.8)	40.9 (6.3)	0.769
Lupus	26.7 (19.1)	28.0 (19.3)	**21.8 (21.5)**	0.588
Musk	24.5 (8.7)	29.9 (9.2)	25.6 (7.3)	0.112
Sonar	33.0 (14.6)	28.5 (14.1)	**26.4 (15.8)**	0.362

Tabela 8.14: Valores de p obtidos nos testes-t realizados para cada par de algoritmos, usando a função de núcleo sigmóide no conjunto de dados *Credit*.

	EE	OPPB
MySVM	**0.001**	**0.001**
EE	-	0.816

Tabela 8.15: Valores de p obtidos nos testes-t realizados para cada par de algoritmos, usando a função de núcleo sigmóide no conjunto de dados *Diabetes*.

	EE	OPPB
MySVM	**0.000**	**0.000**
EE	-	0.232

Tabela 8.16: Valores de p obtidos nos testes-t realizados para cada par de algoritmos, usando a função de núcleo sigmóide no conjunto de dados *Ionosphere*.

	EE	OPPB
MySVM	**0.020**	**0.007**
EE	-	**0.000**

Tabela 8.17: Valor máximo $f(\alpha^*)$ obtido para a função objectivo por cada um dos algoritmos evolucionários, utilizando a função de núcleo sigmóide.

Conjunto de dados	EE	OPPB
Checkerboard	1086.6 (79.2)	1736.8 (115.2)
Spirals	1262.9 (353.5)	4334.9 (367.2)
Threenorm	231.8 (3.8)	261.7 (4.1)
Credit	-7170.3 (593.6)	-4843.0 (310.6)
Diabetes	-2119.7 (492.5)	-682.0 (218.5)
Ionosphere	-557.3 (97.4)	-175.7 (43.1)
Liver	168.4 (7.1)	187.4 (4.9)
Lupus	42.1 (1.8)	49.4 (1.8)
Musk	181.3 (6.4)	204.5 (5.3)
Sonar	98.4 (3.12)	113.0 (2.7)

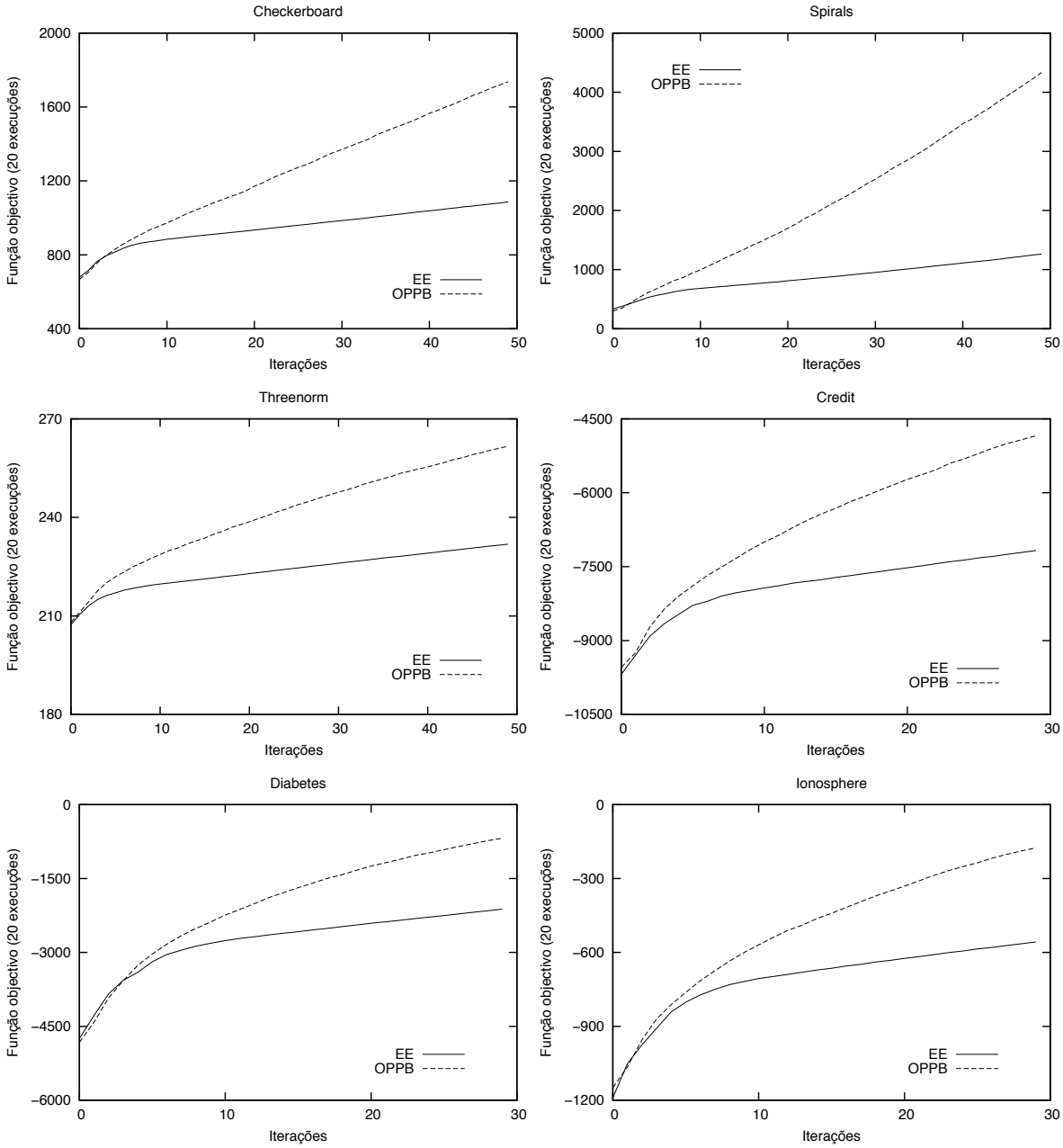


Figura 8.5: Gráficos de convergência da função objectivo para os conjuntos de dados *Checkerboard*, *Spirals*, *Threenorm*, *Credit*, *Diabetes* e *Ionosphere*, utilizando a função de núcleo sigmóide.

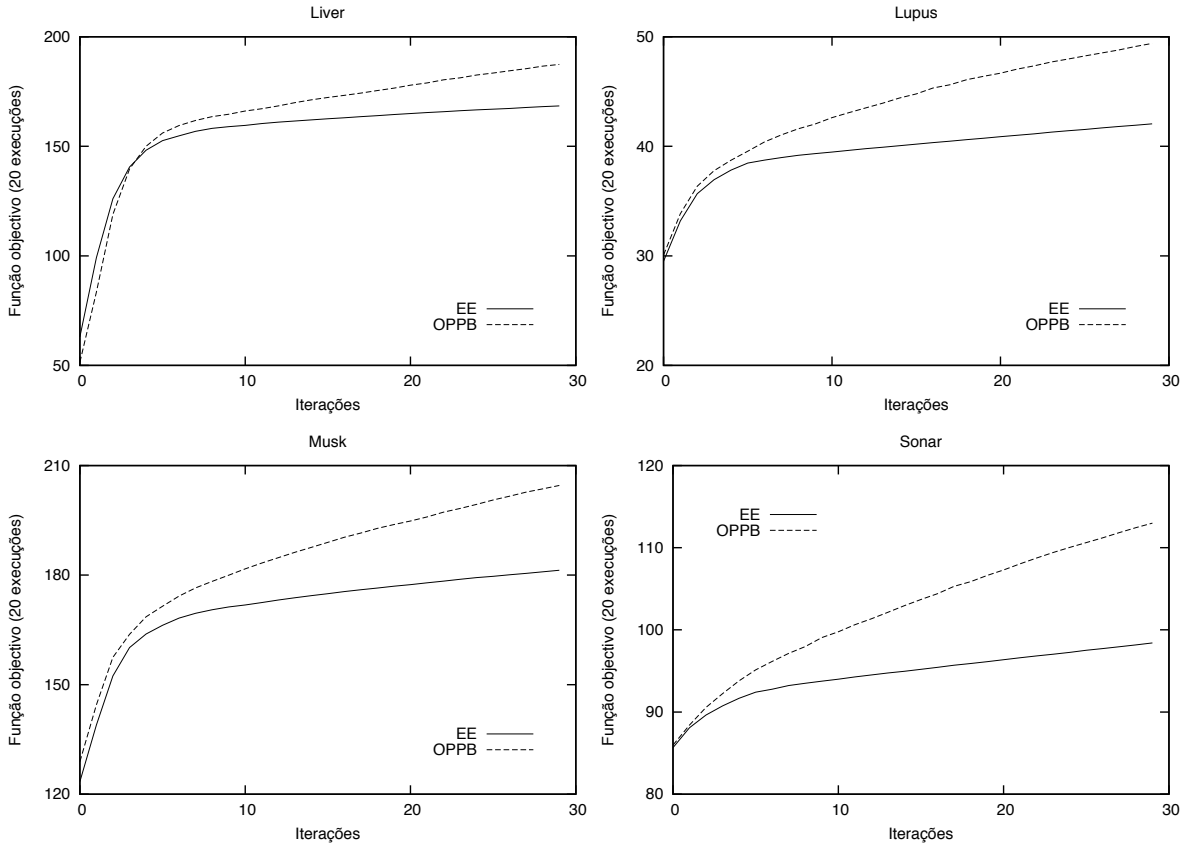


Figura 8.6: Gráficos de convergência da função objectivo para os conjuntos de dados *Liver*, *Lupus*, *Musk* e *Sonar*, utilizando a função de núcleo sigmóide.

Os resultados obtidos nos problemas de optimização, em termos do valor máximo $f(\alpha^*)$ obtido para a função objectivo por cada um dos algoritmos evolucionários, podem ser vistos na tabela 8.23. Tal como nas experiências anteriores, podemos observar que os valores finais da função objectivo são sempre superiores para o OPPB. O facto de termos imposto um limite de iterações bastante reduzido neste conjunto de experiências serviu para exacerbar a diferença de desempenho entre ambos os algoritmos, com vários problemas onde as diferenças são bastante substanciais.

As figuras 8.5 e 8.6 mostram os gráficos de convergência para estes problemas. Os gráficos permitem confirmar as conclusões retiradas a partir dos valores finais, podendo-se observar uma convergência muito mais rápida por parte do algoritmo de enxame. De forma aproximada, podemos afirmar que o OPPB necessita entre 1/2 e 1/5 das iterações utilizadas pela estratégia evolutiva, para atingir o melhor valor obtido por esta. Confirmam-se, assim, os ganhos em termos de complexidade computacional observados já nas experiências anteriores.

No geral, os resultados obtidos nas experiências realizadas com a função de núcleo sigmóide reforçam as conclusões retiradas no conjunto de experiências anterior, onde

foi utilizada a função de núcleo de Epanechnikov. Este reforço é particularmente importante no sentido em que nos permite assumir que as conclusões retiradas após as experiências anteriores, sobre as vantagens do treino evolucionário de MVS, não são específicas à utilização do núcleo de Epanechnikov, podendo-se generalizar a outros núcleos não PSD.

8.4 Funções de Núcleo Resultantes da Substituição da Distância

Neste último conjunto de experiências com núcleos não PSD, investigamos uma situação de maior interesse prático, a qual pode resultar na definição de uma função de núcleo com essas características. Em muitos problemas reais estão disponíveis dados específicos de distância, por exemplo, como resultado de um qualquer processo físico de medição. Nestes casos, uma das possíveis abordagens para a utilização desta informação num método de núcleo, consiste em usar a função de base radial, nela substituindo a distância Euclidiana pelos dados de distância específicos do problema.

Tabela 8.18: Valores de σ_d utilizados para a função de núcleo criada por substituição da medida de distância.

Conjunto de dados	σ_d
Checkerboard	0.006
Spirals	9.684
Threenorm	0.018
Credit	0.002
Diabetes	0.001
Ionosphere	0.247
Liver	0.052
Lupus	0.002
Musk	0.187
Sonar	0.063

Este processo foi formalizado e generalizado para outros núcleos por Haasdonk and Bahlmann [2004]. Quando a função de distância utilizada é isométrica com a norma-$\mathcal{L}^2$, os núcleos resultantes são PSD. Na prática, no entanto, a utilização de medidas de distância específicas, por exemplo distâncias não métricas ou mesmo métricas $\mathcal{L}^p$com $P \neq 2$, pode facilmente resultar em funções de núcleo não PSD [Haasdonk and Bahlmann, 2004].

Para simular esta situação, repetimos as experiências anteriores usando um núcleo FBR onde substituímos a distância Euclidiana pela distância $\mathcal{L}^1$ (frequentemente chamada distância Manhattan), a qual pode produzir matrizes de núcleo não PSD [Haasdonk, 2005]. Ao longo do texto referiremos-nos a este núcleo como função de núcleo resultante da substituição da distância (RSD). As condições experimentais são semelhantes às anteriores, com excepção do número de iterações limite, o qual foi fixado em 300 para os conjuntos de dados sintéticos e 200 para os restantes.

$$k(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(\frac{-\|\mathbf{x}_i - \mathbf{x}_j\|_1^2}{\sigma_d}\right) \tag{8.4}$$

Tabela 8.19: Precisão dos classificadores, em percentagem de erro, utilizando a função de núcleo RSD.

Conjunto de dados	MySVM	EE	OPPB	p
Checkerboard	4.5 (2.4)	5.8 (4.1)	4.7 (3.5)	0.508
Spirals	48.6 (0.9)	49.2 (1.5)	48.8 (1.2)	0.280
Threenorm	20.4 (6.7)	15.0 (6.6)	**14.0 (8.9)**	**0.019**
Credit	30.3 (5.9)	30.0 (5.3)	29.5 (7.7)	0.932
Diabetes	29.3 (6.4)	29.7 (6.5)	27.6 (8.0)	0.618
Ionosphere	32.8 (5.6)	15.7 (8.6)	**12.4 (9.4)**	**0.000**
Liver	40.6 (4.5)	37.7 (5.1)	**36.3 (6.5)**	**0.041**
Lupus	30.2 (20.3)	27.0 (18.1)	28.5 (17.7)	0.862
Musk	45.6 (5.4)	43.5 (2.4)	43.5 (2.4)	0.103
Sonar	12.0 (8.8)	11.0 (9.0)	11.5 (9.6)	0.943

Os resultados referentes à precisão dos classificadores obtidos para o núcleo RSD, apre-

sentados na tabela 8.19, vão de encontro ao que já encontrámos nas experiências realizadas com os núcleos anteriores. Podemos observar que existe diferença com significância estatística nos resultados de três conjuntos de dados, *Threenorm*, *Ionosphereand* e *Liver*. Os resultados dos testes-t (tabelas 8.20, 8.21 e 8.22) mostram que também aqui, nos três casos, as abordagens evolucionárias, em especial o OPPB, têm melhor desempenho do que o algoritmo clássico.

Tabela 8.20: Valores de p obtidos nos testes-t realizados para cada par de algoritmos, usando a função de núcleo RSD, no conjunto de dados *Threenorm*.

	EE	OPPB
MySVM	**0.015**	**0.016**
EE	-	0.692

Tabela 8.21: Valores de p obtidos nos testes-t realizados para cada par de algoritmos, usando a função de núcleo RSD, no conjunto de dados *Ionosphere*.

	EE	OPPB
MySVM	**0.000**	**0.000**
EE	-	0.375

Tabela 8.22: Valores de p obtidos nos testes-t realizados para cada par de algoritmos, usando a função de núcleo RSD, no conjunto de dados *Liver*.

	EE	OPPB
MySVM	0.075	**0.020**
EE	-	0.435

Um aspecto adicional destes resultados, com relevância para o nosso trabalho, tem a ver com o facto de, para dois conjuntos de dados (*Checkerboard* e *Sonar*), ser possível, com esta função de núcleo, obter melhores resultados do que os que obtivemos anteriormente com o núcleo RBF. Embora as diferenças não sejam muito substanciais, não deixam

de indicar a utilidade potencial da utilização de núcleos que normalmente, por não serem positivos definidos, nem sequer são considerados como possibilidade aquando da escolha da função de núcleo para determinado problema.

Embora os resultados obtidos para os algoritmos evolucionários, em termos da precisão de classificação, não sejam muito diferentes (com excepção de dois problemas em que o algoritmo de enxame tem claramente desempenho superior) continua a haver diferenças substanciais em termos dos problemas de optimização. Como podemos observar na tabela 8.23, os valores máximos obtidos para a função objectivo no limite de iterações pelo OPPB continuam a ser consistentemente superiores. Nestas experiências, também é possível observar que, em quase todos os problemas, o desvio padrão é inferior para o algoritmo de enxame, indicando assim uma maior robustez deste, quando comparado com a estratégia evolutiva.

Tabela 8.23: Valor máximo $f(\alpha^*)$ obtido para a função objectivo por cada um dos algoritmos evolucionários, utilizando a função de núcleo RSD.

Conjunto de dados	EE	OPPB
Checkerboard	45.6 (11.1)	135.9 (2.8)
Spirals	411.5 (3.1)	443.9 (1.9)
Threenorm	202.3 (1.3)	209.2 (0.7)
Credit	235.0 (5.0)	269.6 (1.8)
Diabetes	211.7 (5.3)	269.8 (3.8)
Ionosphere	105.5 (1.7)	117.8 (0.9)
Liver	147.0 (1.5)	159.5 (0.5)
Lupus	40.9 (1.1)	43.3 (1.4)
Musk	199.6 (2.4)	218.5 (1.1)
Sonar	57.4 (1.3)	59.8 (0.8)

Os gráficos de convergência, apresentados nas figuras 8.7 e 8.8, ilustram de novo claramente a maior eficiência computacional do algoritmo de enxame. Tal como nos conjuntos de experiências anteriores, podemos observar uma maior rapidez de convergência deste algoritmo, quando comparado com a EE, a qual resulta num menor número de iterações necessárias para produzir classificadores com a mesma qualidade.

Neste conjunto de experiências, podemos observar que normalmente metade das iterações disponíveis (e em vários problemas muito menos) são suficientes para o OPPB atingir os mesmos valores da função de avaliação que a EE obtém no fim da execução.

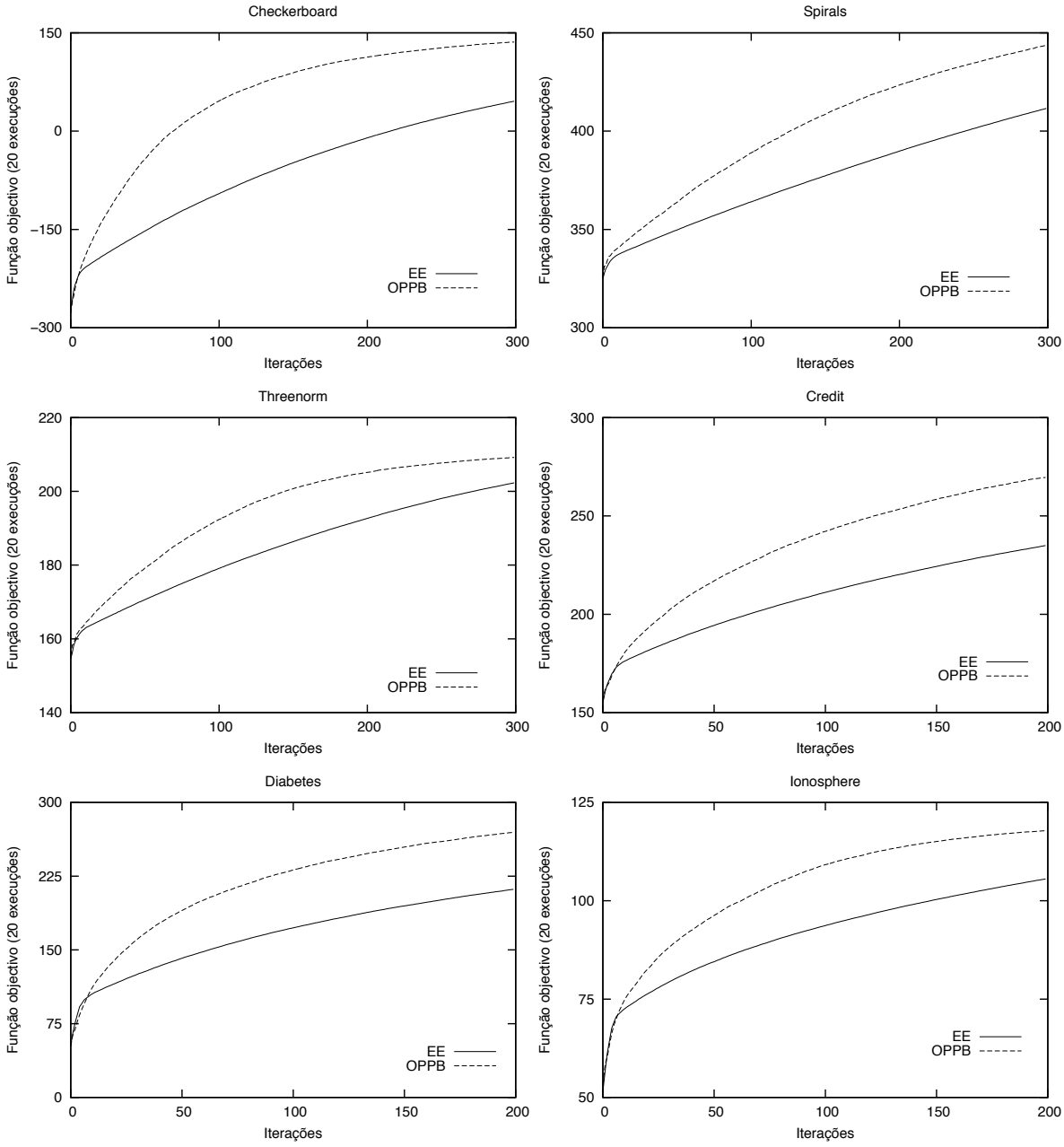


Figura 8.7: Gráficos de convergência da função objectivo para os conjuntos de dados *Checkerboard*, *Spirals*, *Threenorm*, *Credit*, *Diabetes* e *Ionosphere*, utilizando o núcleo RSD.

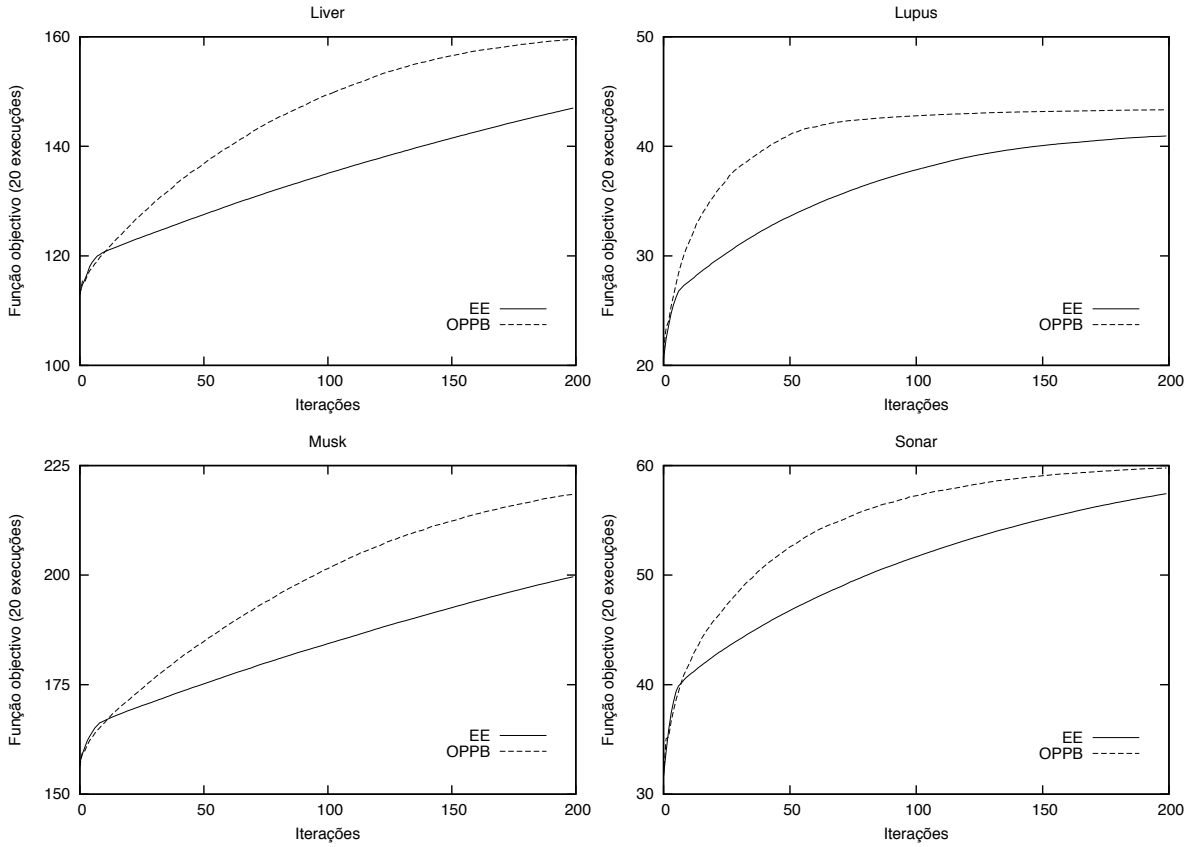


Figura 8.8: Gráficos de convergência da função objectivo para os conjuntos de dados *Liver*, *Lupus*, *Musk* e *Sonar*, utilizando o núcleo RSD.

Este novo conjunto de experiências, além de comprovar todos resultados já descritos para os núcleos anteriores, permite ainda ilustrar as vantagens do treino de MVS num contexto relativamente mais prático, i.e., quando, a partir de medições reais ou outras medidas de distância não Euclideanas, se pretende construir um núcleo por substituição da medida de distância numa função de núcleo bem conhecida. Neste caso, verificámos que, para o núcleo resultante da substituição, os algoritmos evolucionários, e em especial o OPPB, conseguiram obter resultados significativamente melhores, em vários dos problemas do conjunto de teste.

8.5 Resultados da Optimização de MVS

As experiências anteriores permitiram validar o interesse da utilização de núcleos não PSD em máquinas de vectores de suporte, bem como a utilização do algoritmo por nós proposto para o seu treino. Nesta secção, testamos o algoritmo integrado de optimização de máquinas de vectores de suporte, o qual utiliza a abordagem ao treino

baseada em inteligência no seu nível interior, i.e., para treinar as MVS correspondentes aos núcleos e parâmetros codificados nas partículas do algoritmo principal.

Nas experiências realizadas, fornecemos cada um dos conjuntos de dados a um novo operador do RapidMiner que implementa a abordagem descrita. O algoritmo exterior foi executado durante 1000 iterações, com um enxame de 20 partículas para cada problema, sendo devolvido o vector correspondente à melhor solução encontrada (núcleo e parâmetros). A avaliação de cada partícula foi feita utilizando o OPPB de treino de MVS, ao qual foi permitido utilizar 500 iterações com 20 partículas para treinar o núcleo. Para promover uma melhor avaliação do núcleo, esta foi também realizada por validação cruzada 20-*fold*. Os resultados de precisão que reportamos resultam de uma validação cruzada 20-*fold* final, realizada após o término do algoritmo, utilizando a melhor partícula por ele devolvida.

Realizámos dois conjuntos de experiências para testar a nossa abordagem. No primeiro conjunto, cujos resultados são apresentados na tabela 8.24, optimizámos todos os parâmetros, com excepção do C, de maneira a permitir comparar os resultados com os obtidos no primeiro conjunto de experiências. Relembramos que aí tentámos obter os melhores resultados para cada problema utilizando o que se pode considerar a abordagem clássica: treino com algoritmo baseado em programação quadrática, núcleo RBF e optimização em grelha para obter os parâmetros que produzem o melhor classificador. No entanto não optimizámos C, o qual foi fixado a 1, tal como neste novo conjunto de experiências.

No segundo conjunto de experiências, incluímos o parâmetro C nas partículas, procurando investigar se a sua optimização levaria à obtenção de melhores classificadores. Adicionalmente, estas experiências permitem determinar se a abordagem proposta funciona na sua totalidade, i.e., optimizando não só os parâmetros de núcleo, mas também o valor de C. Os resultados destas experiências estão listados na tabela 8.25. Note-se que, devido à codificação utilizada, para os parâmetros w_i e C são apresentados os valores de $\log_{10}$ de cada um. A precisão do classificador é medida pela percentagem de instâncias correctamente classificadas, sendo que os resultados apresentados resultam de uma única execução do algoritmo. A análise dos resultados obtidos em ambos os conjuntos de experiências permite a constatação de vários factos:

- O facto mais significativo é que a abordagem proposta é claramente viável, no sentido em que foi capaz de produzir núcleos que resultam numa separação das classes conducente à obtenção de resultados pelo menos tão bons como os obtidos pelas abordagens clássicas (ver tabela 8.2).
- Para vários problemas, os núcleos obtidos resultam numa classificação mais precisa do que a obtida pela abordagem clássica. De especial relevo são os resultados para os conjuntos de dados *Lupus* e *Sonar*, onde as diferenças são substanciais.

- Quando comparados com os resultados obtidos pelos diversos núcleos das experiências anteriores, podemos constatar que os obtidos nestas experiências são sempre semelhantes ou superiores.

- Comparando os resultados obtidos com e sem a optimização de C, não se nota diferença que mereça menção. O algoritmo mostrou-se capaz de encontrar combinações de C e dos restantes parâmetros que resultam num desempenho semelhante ao obtido nas experiências anteriores, mesmo com o espaço de procura alargado.

- Existem vários núcleos nos resultados para os quais podemos observar FBRs com pesos negativos. Estas funções de núcleo podem ser não PSD, o que ilustra a capacidade desta abordagem em lidar com núcleos com essas características.

Tabela 8.24: Parâmetros do melhor núcleo encontrado para cada problema e precisão da máquina de vectores de suporte correspondente.

Conj. de Dados	$\log_{10}(\sigma_1)$	$\log_{10}(\sigma_2)$	$\log_{10}(\sigma_3)$	w_1	w_2	w_3	Precisão
Checkerboard	-2.805	3.000	-1.002	1.000	0.366	1.000	95.10 (3.49)
Spirals	-1.959	-0.182	2.591	1.000	0.0125	0.385	99.80 (0.87)
Threenorm	-0.973	-0.183	0.357	0.401	0.293	1.000	87.80 (4.98)
Credit	-0.077	2.193	1.075	0.171	0.777	0.691	86.66 (7.22)
Diabetes	0.187	-0.501	3.00	0.898	-0.325	0.414	78.39 (6.40)
Ionosphere	0.729	-2.416	2.736	0.422	-0.238	0.723	95.46 (5.21)
Liver	0.120	0.403	1.025	0.381	0.494	-0.046	72.99 (8.13)
Lupus	0.721	-1.132	2.785	0.824	-0.115	0.986	81.25 (17.88)
Musk	-0.187	0.541	1.508	-0.008	0.950	1.000	93.89 (4.56)
Sonar	-0.286	0.455	0.637	0.121	0.775	0.533	91.36 (8.84)

Tabela 8.25: Parâmetros do melhor núcleo encontrado para cada problema (incluindo C) e precisão da máquina de vectores de suporte correspondente.

Conj. de Dados	$log_{10}(C)$	$\log_{10}(\sigma_1)$	$\log_{10}(\sigma_2)$	$\log_{10}(\sigma_3)$	w_1	w_2	w_3	Precisão
Checkerboard	2.820	-2.582	-1.385	-1.504	0.565	0.217	0.066	94.60 (3.23)
Spirals	-0.125	-1.898	-0.550	-2.818	1.000	0.986	0.225	99.80 (0.87)
Threenorm	1.850	-0.605	0.273	0.017	0.538	0.802	0.040	87.80 (6.26)
Credit	0.285	0.233	-2.048	-2.423	0.513	-1.000	-0.614	86.39 (5.10)
Diabetes	0.761	2.014	-1.343	0.640	0.807	0.118	0.232	76.96 (7.24)
Ionosphere	0.965	0.579	2.158	2.351	0.123	0.847	-0.314	96.00 (3.67)
Liver	0.750	0.781	2.560	2.465	0.592	-0.521	0.395	72.68 (10.81)
Lupus	1.141	0.591	1.173	2.561	0.777	0.974	0.665	80.75 (15.91)
Musk	0.299	0.939	-1.034	2.511	0.748	0.038	0.367	94.73 (4.76)
Sonar	1.174	0.414	0.824	1.151	0.823	0.902	-0.187	90.36 (9.31)

De forma geral, podemos assim concluir que a abordagem proposta permite obter classificadores de elevada qualidade para os problemas de teste, sem haver necessidade de intervenção do utilizador na escolha do núcleo, dos seus parâmetros ou do valor de C. Como dissemos, no entanto, este algoritmo é sobretudo uma prova de conceito, já que, embora mostre que a abordagem é viável, apenas permite a exploração de um espaço de núcleos limitado à combinação linear de várias FBR. Mesmo assim, ficou demonstrada a sua utilidade, já que, mesmo neste conjunto limitado de problemas de teste, obteve sempre resultados competitivos, e mesmo superiores aos da abordagem clássica em dois problemas.

Apesar da limitação inerente à solução apresentada, a abordagem geral é genérica e pode ser facilmente extensível a espaços de procura mais interessantes. A separação da representação num triplo $\langle C, \mathbf{p}, x\rangle$, no qual C é o parâmetro de regularização, $\mathbf{p}$ é um vector de parâmetros numéricos e x é uma estrutura correspondendo à função de núcleo propriamente dita, permite facilmente imaginar algoritmos evolucionários capazes de evoluir funções de núcleo bastante mais diversificadas.

Uma hipótese que se apresenta imediatamente, envolve a utilização de programação genética para evoluir a componente x da representação, à imagem do que é feito, por exemplo, por Howley and Madden [2005]. A programação genética permitiria evoluir funções de núcleo arbitrárias a partir de vários tipos de primitivas, que poderiam incluir funções de núcleo já bem conhecidas, bem como medidas de distância (ou proximidade)

provenientes, por exemplo, de processos físicos ou do pré-processamento dos dados.

Os parâmetros numéricos poderiam continuar a ser evoluídos por um OPPB, sendo a coordenação feita através de uma estratégia de co-evolução. Uma vantagem adicional desta ideia seria que, neste caso, p poderia ser utilizado para encontrar os terminais numéricos da função (pesos, parâmetros, constantes...).

Uma abordagem deste tipo teria duas vantagens substanciais sobre abordagens anteriores que evoluem funções de núcleo. Por um lado, trata-se de uma abordagem integrada que evolui conjuntamente a função, o parâmetro da MVS e ainda outros parâmetros necessários. Por outro lado, trata-se da primeira abordagem a utilizar um algoritmo de treino capaz de lidar com núcleos não PSD. Entre as possíveis desvantagens está o facto de ser necessário alguma intervenção do utilizador na escolha de funções e terminais a fornecer ao algoritmo de programação genética. A complexidade computacional, essencialmente em termos de tempo, também aumentaria com a utilização de programação genética.

Capítulo 9

Conclusões e Trabalho Futuro

Neste capítulo sumariamos as contribuições mais substanciais do nosso trabalho, tanto para a área da inteligência de enxame, como para a integração desta com os métodos de núcleo, mais especificamente as máquinas de vectores de suporte. Com base nos resultados experimentais obtidos retiramos algumas conclusões globais e resumimos as conclusões apresentadas ao longo da dissertação. Finalmente, descrevemos as linhas maiores de investigação futura que se nos apresentam após a conclusão deste trabalho.

9.1 Conclusões

Esta dissertação teve como objecto a intersecção entre duas áreas científicas de grande sucesso: as máquinas de vectores de suporte, elemento emblemático do grupo de métodos de núcleo, e a computação evolucionária. Um dos contributos nela apresentados consiste num estado da arte detalhado dessa intersecção, onde são analisadas as abordagens evolucionárias mais significativas a aspectos das MVS como a selecção de características, a optimização de parâmetros, a síntese de núcleos e o seu treino. Terminámos esse estado da arte com a identificação de algumas áreas onde, na nossa opinião podíamos fazer contribuições significativas, nomeadamente ao nível dos algoritmos de treino evolucionários e da integração de optimização de diferentes aspectos.

Uma parte substancial do trabalho realizado, discutido detalhadamente nos capítulos correspondentes, centrou-se no desenvolvimento de um algoritmo de enxame, que, embora de uso geral, tivesse características que permitissem a sua aplicação eficaz ao treino de MVS. Denominámos esse algoritmo optimizador predador-presa com batedores. Na parte final da dissertação apresentámos uma versão do OPPB especialmente adaptada ao treino de máquinas de vectores de suporte. O algoritmo procura beneficiar das características dos batedores escolhidos especificamente para esta tarefa, de

maneira a obter um desempenho no treino pelo menos comparável, em termos do problema de optimização, com outras abordagens evolucionárias e, em termos da precisão dos classificadores finais, com as abordagens baseadas em programação quadrática.

Começámos por demonstrar, num conjunto de problemas de teste selecionados para o efeito, que o OPPB consegue produzir classificadores de qualidade semelhante aos treinados por algoritmos clássicos e pela melhor abordagem evolucionária conhecida. Esta é a primeira instância que conhecemos de um algoritmo de enxame capaz de o fazer, já que tentativas anteriores com OEP básicos pareciam produzir soluções sub-óptimas, facto que comprovámos adicionando um OEP padrão ao nosso ambiente experimental.

A análise dos gráficos de convergência dos problemas de optimização correspondentes a estas primeiras experiências permitiu-nos observar que, não só o OPPB era competitivo com a melhor abordagem evolucionária, como conseguia atingir valores semelhantes da função objectivo, necessitando de substancialmente menos iterações (e, consequentemente, menos avaliações da função objectivo) do que o algoritmo baseado numa estratégia evolutiva.

Este resultado foi posteriormente confirmado comparando as abordagens evolucionárias em vários núcleos não PSD, os quais podem produzir problemas multimodais. Em todas as experiências realizadas, para todas as combinações núcleo / problema, o algoritmo de enxame necessitou de substancialmente menos iterações (tipicamente de 1/2 a 1/5) para atingir os mesmos valores da função objectivo atingidos pela EE no limite de iterações. Estes resultados estabelecem o OPPB, não só como o primeiro algoritmo de enxame a ser utilizado no treino de MVS com núcleos não PSD, mas também como a abordagem evolucionária mais eficiente nessa tarefa.

A eficiência no problema de optimização, só por si, não traduz o interesse prático da nossa abordagem. Esse interesse está essencialmente no facto de as abordagens evolucionárias, sendo algoritmos de optimização global, permitirem, em teoria, lidar melhor com os problemas de optimização multi-modais resultantes da utilização de núcleos não PSD, do que as abordagens baseadas em programação quadrática. Esta característica, se confirmada, permitiria utilizar com mais confiança núcleos não PSD desenvolvidos para estruturas complexas, não vectoriais, ou resultantes de medidas de distância (ou proximidade) úteis, mas que resultam em núcleos não PSD.

Neste trabalho estendemos os resultados conhecidos da utilização de abordagens evolucionárias no treino de MVS com núcleos não PSD, os quais, até agora, se limitavam ao uso de estratégias evolutivas com o núcleo de Epanechnikov. Realizámos experiências com dois outros núcleos não PSD, o núcleo sigmóide e um núcleo resultante da substituição da medida de distância no núcleo FBR. Em todos os conjuntos de experiências realizados encontrámos pares problema / núcleo para os quais as abordagens evolucionárias produziram classificadores com um desempenho superior ao da abordagem

clássica considerada (MySVM).

Tão importante como obter resultados superiores em alguns problemas, foi o facto do algoritmo de treino baseado em partículas não ter produzido classificadores com um desempenho pior (com significância estatística) do que a abordagem baseada em programação quadrática em nenhum par problema / núcleo de todas as experiências realizadas. O mesmo não pode ser dito da abordagem baseada numa estratégia evolutiva, mas uma análise dos gráficos de convergência permite determinar que este facto resulta das limitações por nós impostas ao número máximo de iterações disponíveis. Tendo em atenção as curvas desses gráficos, pode-se concluir que, com recursos computacionais adicionais, a EE poderia produzir classificadores com qualidade semelhante aos resultantes da utilização do OPPB.

Estes resultados confirmam e reforçam a importância que as abordagens evolucionárias podem ter quando, para determinado problema, um utilizador é obrigado a considerar um núcleo não PSD (ou um núcleo para o qual é difícil provar essa propriedade). A utilização destes métodos com o núcleo resultante da substituição da medida de distância no núcleo FBR, em particular, demonstra como os algoritmos evolucionários podem ser úteis num contexto prático, i.e., quando um mecanismo comum de criação de núcleos (a substituição da medida de distância) é utilizado num novo problema.

Uma das dificuldades de utilização de máquinas de vectores de suporte por um utilizador que não seja experiente na área tem a ver com a escolha de uma função de núcleo apropriada para o problema, bem como os seus parâmetros, além do parâmetro C da MVS. Propusemos uma abordagem totalmente baseada em inteligência de enxame, a qual usa dois OPPB diferentes para, por um lado, escolher o núcleo e os seus parâmetros e, por outro, treinar as máquinas de vectores de suporte correspondentes a cada partícula do primeiro OPPB, as quais podem produzir um núcleo não PSD.

Apesar desta abordagem estar limitada a núcleos resultantes da combinação linear de FBR, ficou demonstrada a sua utilidade prática quando os exemplos de treino são vectores reais. Não só o algoritmo foi capaz de encontrar núcleos e parâmetros que permitiram encontrar classificadores tão bons como os obtidos utilizando um algoritmo clássico com o núcleo FBR e parâmetros escolhidos por procura em grelha, como também, para dois dos problemas, produziu núcleos que resultaram nos melhores classificadores treinados em todas as experiências realizadas.

O algoritmo proposto é assim a primeira abordagem integrada de optimização de MVS, totalmente baseada em algoritmos de enxame e capaz de lidar com núcleos não PSD. O algoritmo é capaz de produzir um novo núcleo e respectivos parâmetros para um problema fornecido pelo utilizador, sem intervenção deste, e desde que os exemplos sejam constituídos por atributos numéricos. Discutimos ainda a ultrapassagem desta limitação através da utilização de programação genética para generalizar o domínio de

núcleos produzidos, bem como o tipo de dados aceites pelo algoritmo.

De um ponto de vista mais prático, note-se ainda que todos os algoritmos de treino, bem como o algoritmo de optimização de MVS, foram implementados como operadores do software RapidMiner. Sendo este um pacote de uso comum, tanto na comunidade de aprendizagem, como de análise inteligente da dados, os operadores podem facilmente ser utilizados, tanto para reproduzir os nossos resultados, como para tirar partido das funcionalidades que implementam.

9.2 Trabalho Futuro

Dos resultados do trabalho que deu origem a esta dissertação emergem duas avenidas de investigação futura, relacionadas com as áreas em que esse trabalho foi realizado. Do ponto de vista da computação evolucionária, ou, mais especificamente da inteligência de enxame, o algoritmo de optimização predador-presa com batedores é um algoritmo de optimização genérico com grande potencialidade de aplicação a áreas diversificadas. Esta potencialidade é desde logo sublinhada pela mais de uma centena de citações de que foram alvo os artigos que originalmente apresentaram o mecanismo predador-presa, incluindo diversos de aplicação, melhoramento e generalização do princípio básico. Também para a utilização de partículas batedoras, embora de introdução mais recente, já se encontram novas aplicações na bibliografia da área. Esta popularidade dos mecanismos introduzidos no algoritmo de enxame descrito nesta dissertação sublinha a sua utilidade.

Em relação à optimização de MVS por inteligência de enxame (e outros métodos evolucionários), permanecem em aberto várias linhas de investigação, algumas já identificadas anteriormente:

- Neste trabalho não foi feito nenhum esforço de optimização do algoritmo de treino, e.g., do ponto de vista paramétrico ou do estudo de condições de paragem alternativas, de maneira a melhorar o desempenho do algoritmo. Esta questão é importante, já que a eficiência do algoritmo é essencial tanto para a sua utilização no algoritmo de optimização de MVS, como para problemas de maior dimensão, situações em que o custo computacional da abordagem é particularmente relevante.

- A questão da aplicação a problemas de maior dimensão é também interessante. Um aspecto particular do problema de optimização decorrente do treino de uma MVS é o facto do número de dimensões corresponder ao número de instâncias do conjunto de treino. Esta elevada dimensionalidade impede inclusivamente a utilização de algoritmos evolucionários de grande sucesso, como a EE-AMC. Nas

nossas experiências utilizámos problemas com um máximo de 1000 instâncias, não se tendo verificado um deterioramento significativo do desempenho do algoritmo. A forma como o algoritmo escala para outras magnitudes do número de instâncias merece investigação adicional, já que muitos problemas de interesse prático podem possuir dezenas ou centenas de milhares de instâncias.

- Nesta dissertação apresentámos um algoritmo de optimização de MVS com algumas limitações, tanto em relação ao tipo de instâncias, como aos núcleos produzidos. Em relação aos aspectos optimizados, deixámos de fora a escolha dos atributos ou características passíveis de utilização pelo algoritmo de classificação. Uma primeira possibilidade de expansão da abordagem apresentada poderia consistir na integração deste elemento no algoritmo de optimização. Como a abordagem típica consiste em considerar um atributo binário por atributo, seria possível continuar uma abordagem totalmente baseada em inteligência de enxame à optimização de MVS incluindo selecção de características.

- As outras possibilidades de expansão a esta abordagem, nomeadamente no que diz respeito à utilização de instâncias não vectoriais ou heterogéneas e à geração de núcleos mais genéricos implicariam a utilização de programação genética e uma maior intervenção por parte do utilizador em termos de fornecimento das funções primitivas e terminais. Embora constitua uma abordagem mais complexa, é também uma das linhas de investigação mais promissora, tanto pelos resultados interessantes já obtidos utilizando programação genética, como também pelas possibilidades inerentes à integração, numa abordagem deste tipo, dum algoritmo de treino como o proposto, capaz de treinar núcleos não PSD.

- Finalmente, a utilização feita neste trabalho, tanto do algoritmo de treino, como de optimização de MVS, cingiu-se a problemas de teste comuns na bibliografia da área. Um passo natural desta investigação será, obviamente a identificação de problemas de interesse prático e imediato, onde a aplicação da mesma possa trazer contributos significativos.

Apêndice A

Funções de Teste

A.1 Esfera

A função esfera, também conhecida por primeira função de *De Jong*, é uma das funções de teste mais simples, sendo contínua, convexa, unimodal e separável. O espaço de procura é limitado ao hiper-cubo $-100 \leqslant x_i \leqslant 100, i = 1, ..., n$, com o óptimo global $f(\mathbf{x}^*) = 0$ para $\mathbf{x}^* = \mathbf{0}$. A figura A.1 apresenta uma representação gráfica desta função a 2 dimensões.

$$f_1(\mathbf{x}) = \sum_{i=1}^{n} x_i^2 \tag{A.1}$$

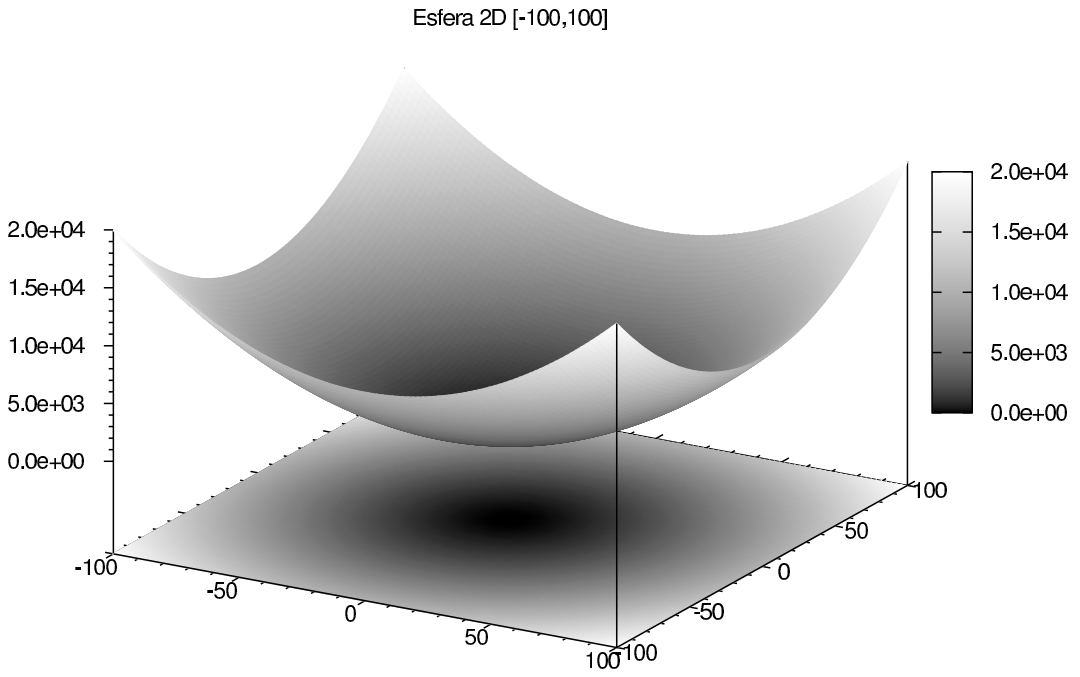


Figura A.1: Representação gráfica da função esfera em duas dimensões.

A.2 Elipsóide com Rotação

A função hiper-elipsóide com rotação também possui uma estrutura simples, resultando, para cada par de eixos, numa elipse com rotação. É contínua, convexa e unimodal, sendo, no entanto não-separável, o que, em conjunção com a rotação das elipsóides, traz dificuldades acrescidas à sua optimização. O espaço de procura é limitado ao hipercubo $-100 \leqslant x_i \leqslant 100, i = 1, ..., n$, com o óptimo global $f(\mathbf{x}^*) = 0$ para $\mathbf{x}^* = \mathbf{0}$. A figura A.2 apresenta uma representação gráfica desta função a 2 dimensões.

$$f_2(\mathbf{x}) = \sum_{i=1}^{n} \left(\sum_{j=1}^{i} x_j \right)^2 \tag{A.2}$$

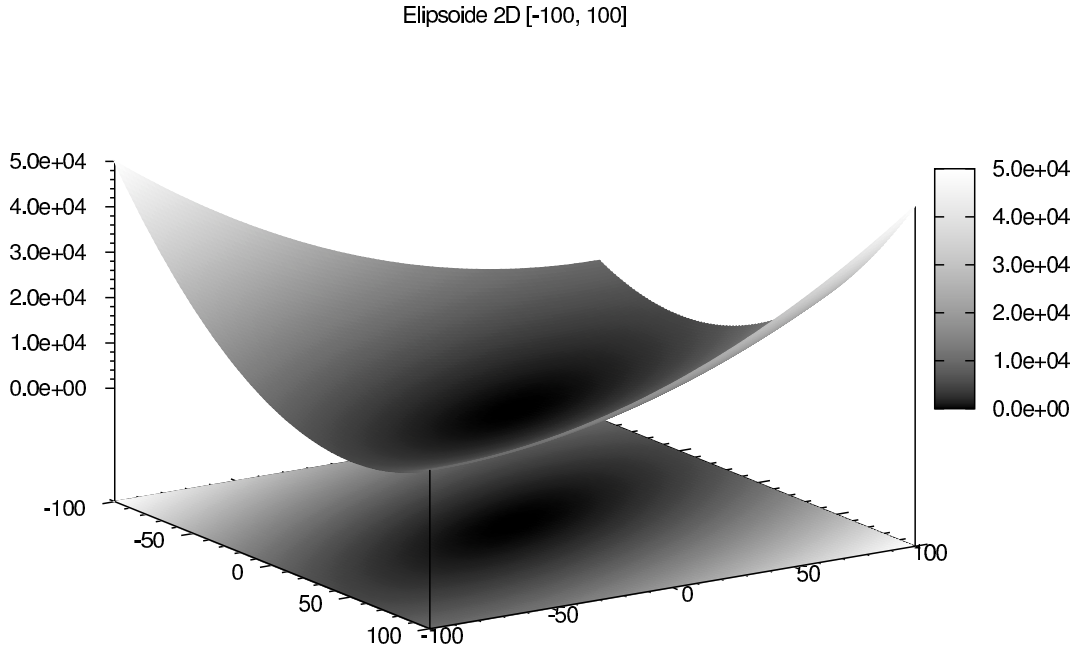


Figura A.2: Representação gráfica da função elipsóide com rotação em duas dimensões.

A.3 Função de Zhakarov

A função de Zhakarov é contínua, convexa, unimodal e separável, mas o óptimo global encontra-se numa zona quase plana onde existe muito pouca informação associada ao gradiente da função para guiar os algoritmos. Esta característica dificulta, obviamente, a tarefa dos algoritmos evolucionários cuja operação dependa dessa informação. O espaço de procura é limitado ao hiper-cubo $-5 \leqslant x_i \leqslant 10, i = 1, ..., n$, com o óptimo global $f(\mathbf{x}^*) = 0$ para $\mathbf{x}^* = \mathbf{0}$. A figura A.3 apresenta uma representação gráfica desta função a 2 dimensões, sendo bem visível a zona plana onde se encontra o óptimo.

$$f_3(\mathbf{x}) = \sum_{i=1}^{n} x_i^2 + \left(\sum_{i=1}^{n} 0.5ix_i\right)^2 + \left(\sum_{i=1}^{n} 0.5ix_i\right)^4 \tag{A.3}$$

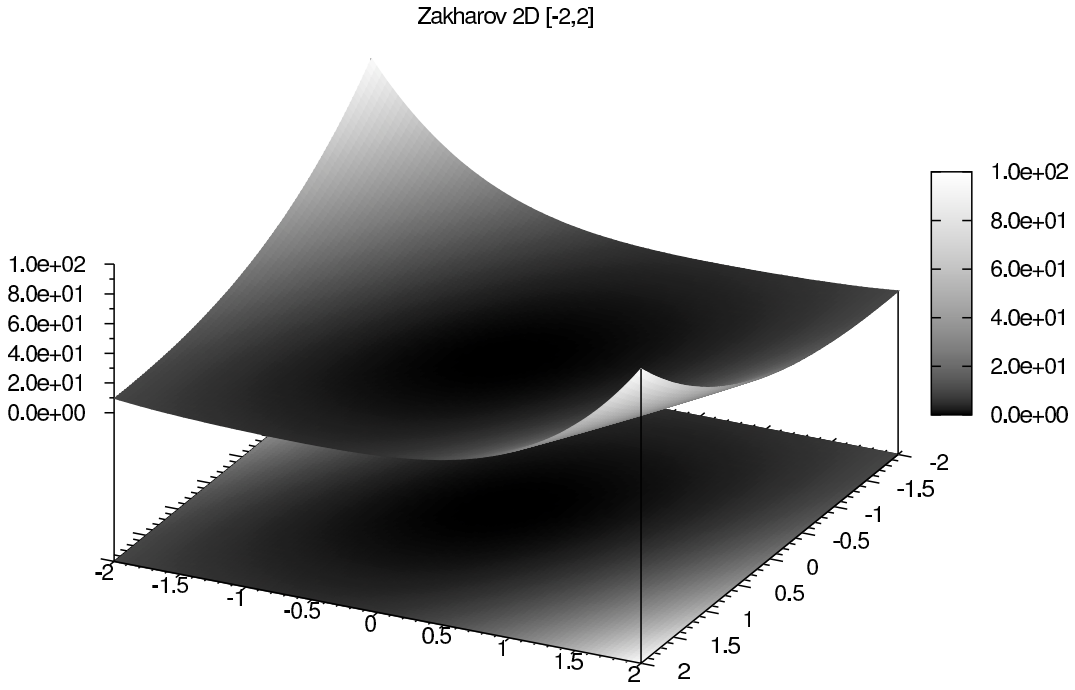


Figura A.3: Representação gráfica da função de Zakharov em duas dimensões.

A.4 Função de Rastrigin

A função de Rastrigin é multimodal e separável. Resulta da adição à função esfera da modulação resultante de um termo com a função coseno, o qual produz um número muito elevado de óptimos locais. A localição destes nos espaço de procura é, no entanto, regular, o que de alguma forma facilita a sua optimização. O espaço de procura é limitado ao hiper-cubo $-5.12 \leqslant x_i \leqslant 5.12, i = 1, ..., n$, com o óptimo global $f(\mathbf{x}^*) = 0$ para $\mathbf{x}^* = \mathbf{0}$. A figura A.4 apresenta uma representação gráfica desta função a 2 dimensões, onde a localização regular dos óptimos locais está claramente ilustrada.

$$f_4(\mathbf{x}) = 10n + \sum_{i=1}^{n} \left(x_i^2 - 10\cos(2\pi x_i)\right) \tag{A.4}$$

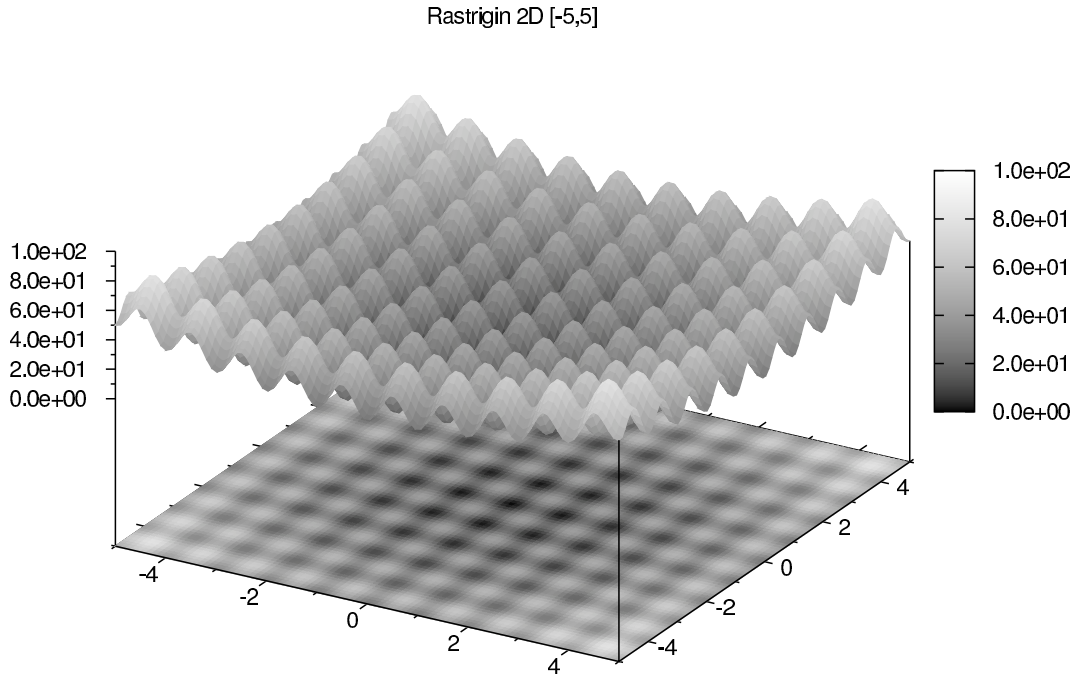


Figura A.4: Representação gráfica da função de Rastragin em duas dimensões.

A.5 Função de Ackley

A função de Ackley apresenta dificuldades semelhantes às da função de Rastrigin, já que, embora a estrutura global da função não seja convexa como no caso de f_5, também nesta função existem muitos mínimos locais dispostos de forma relativamente regular. Esta disposição facilita a transição dos algoritmos entre óptimos, mantendo a dificuldade do problema moderada. O problema é multimodal e separável. O espaço de procura é limitado ao hiper-cubo $-32 \leqslant x_i \leqslant 32, i = 1, ..., n$, com o óptimo global $f(\mathbf{x}^*) = 0$ para $\mathbf{x}^* = \mathbf{0}$. A figura A.5 apresenta uma representação gráfica desta função a 2 dimensões na totalidade do domínio, sendo que na figura A.6 apresentamos um detalhe em torno do óptimo global, no qual é mais claramente visível a estrutura regular de óptimos locais.

$$f_5(\mathbf{x}) = -20 \exp\left(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}\right) - \exp\left(\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi x_i)\right) + 20 + e \qquad \text{(A.5)}$$

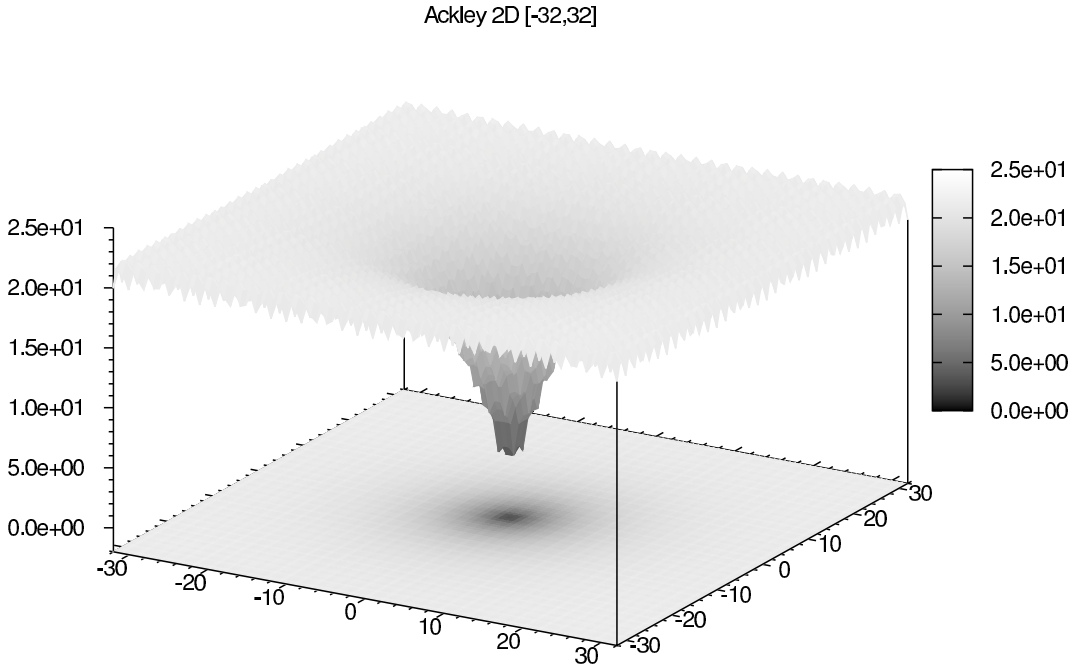


Figura A.5: Representação gráfica da função de Ackley em duas dimensões - vista global.

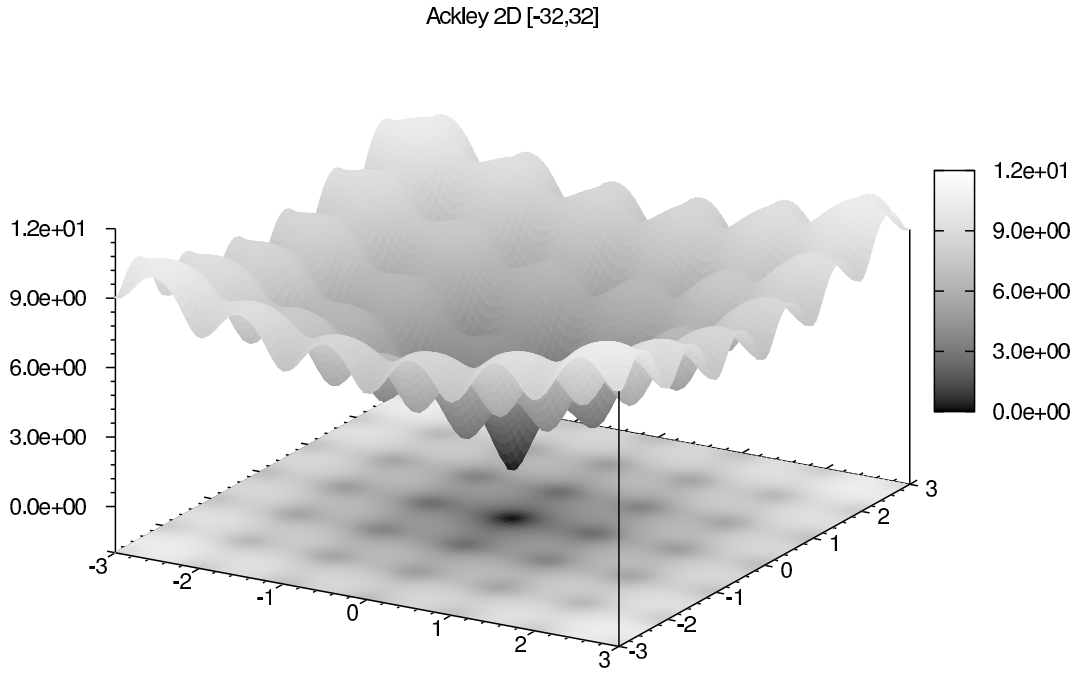


Figura A.6: Representação gráfica da função de Ackley em duas dimensões - detalhe.

A.6 Função de Griewangk

A função de Griewangk é altamente multimodal como as anteriores, possuindo a mesma estrutura global convexa que a função de Rastrigin. No entanto, ao contrário daquela, é não-separável, incluindo um termo que implementa interacções entre os seus parâmetros. Apesar disso, o espaçamento regular dos óptimos facilita a sua optimização. O espaço de procura é limitado ao hiper-cubo $-600 \leqslant x_i \leqslant 600, i = 1, ..., n$, com o óptimo global $f(\mathbf{x}^*) = 0$ para $\mathbf{x}^* = \mathbf{0}$. A figura A.7 apresenta uma representação gráfica desta função a 2 dimensões na totalidade do domínio, ilustrando a sua estrutura global semelhante à função esfera. A observação do detalhe na figura A.8, mostra, no entanto a profusão de óptimos regularmente espaçados.

$$f_6(\mathbf{x}) = \frac{1}{4000}\sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n} \frac{\cos(x_i)}{\sqrt{i}} + 1 \tag{A.6}$$

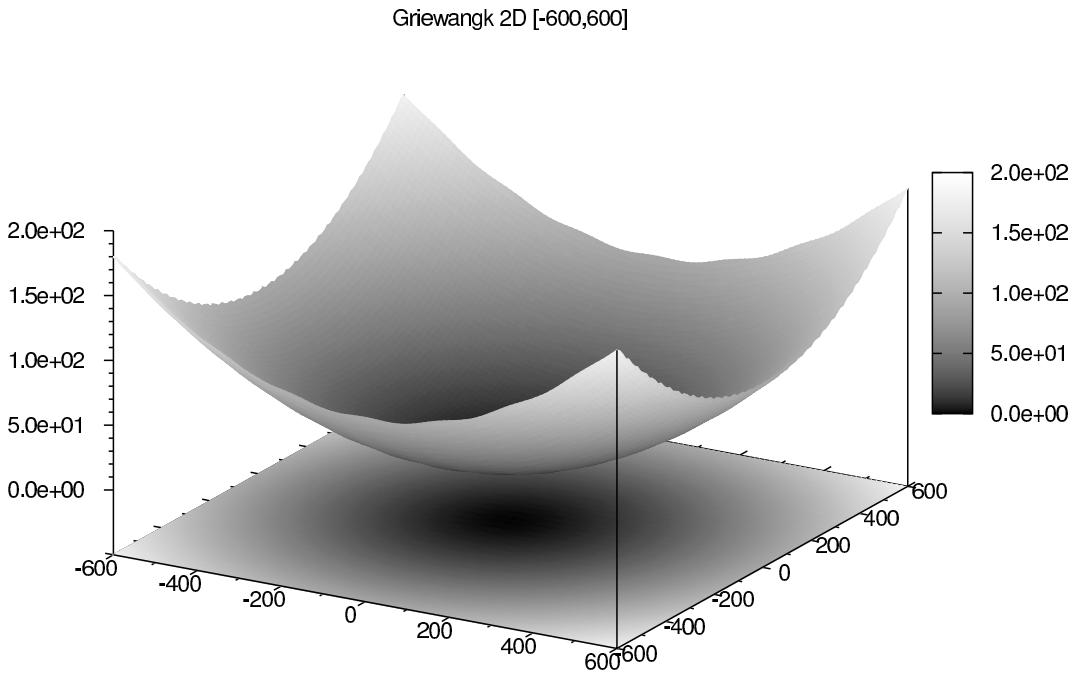


Figura A.7: Representação gráfica da função de Griewangk em duas dimensões - vista global.

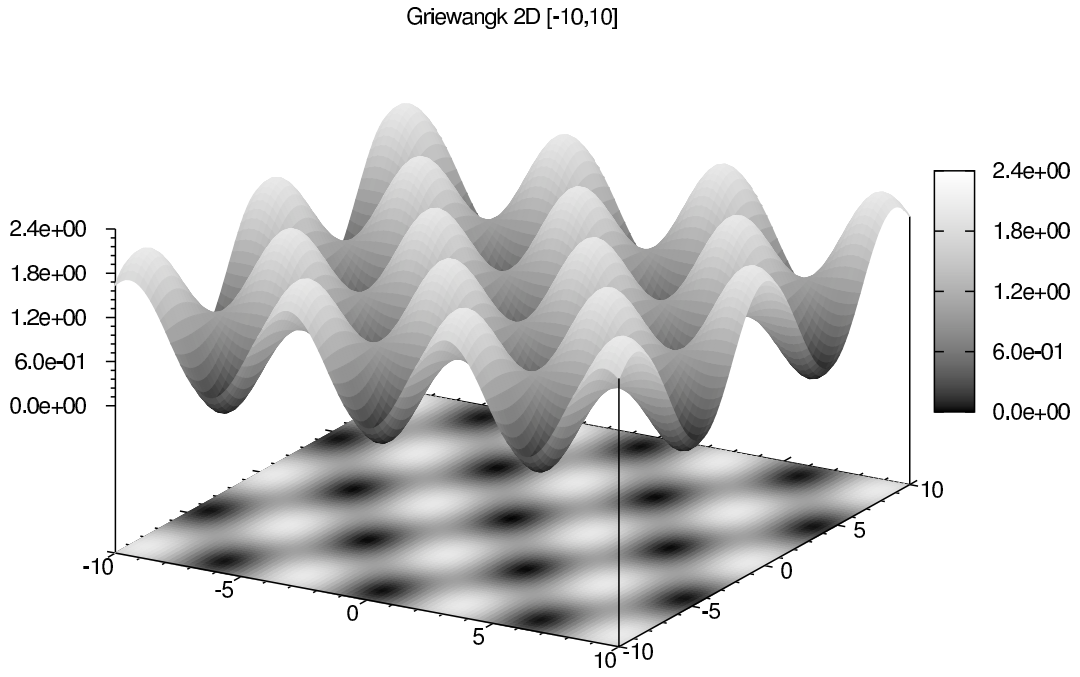


Figura A.8: Representação gráfica da função de Griewangk em duas dimensões - detalhe.

A.7 Função de Salomon

A função de Salomon, tendo algumas características semelhantes às das funções anteriores, acrescenta uma dificuldade adicional à optimização, ao não manter a mesma estrutura regular de óptimos locais. É também não-separável e altamente multimodal. O espaço de procura é limitado ao hiper-cubo $-100 \leqslant x_i \leqslant 100, i = 1, ..., n$, com o óptimo global $f(\mathbf{x}^*) = 0$ para $\mathbf{x}^* = \mathbf{0}$. A figura A.9 apresenta uma representação gráfica desta função a 2 dimensões na totalidade do domínio e a figura A.10 mostra o detalhe na vizinhança do óptimo global.

$$f_7(\mathbf{x}) = 1 - \cos(2\pi\|\mathbf{x}\|) + 0.1\|\mathbf{x}\| \tag{A.7}$$

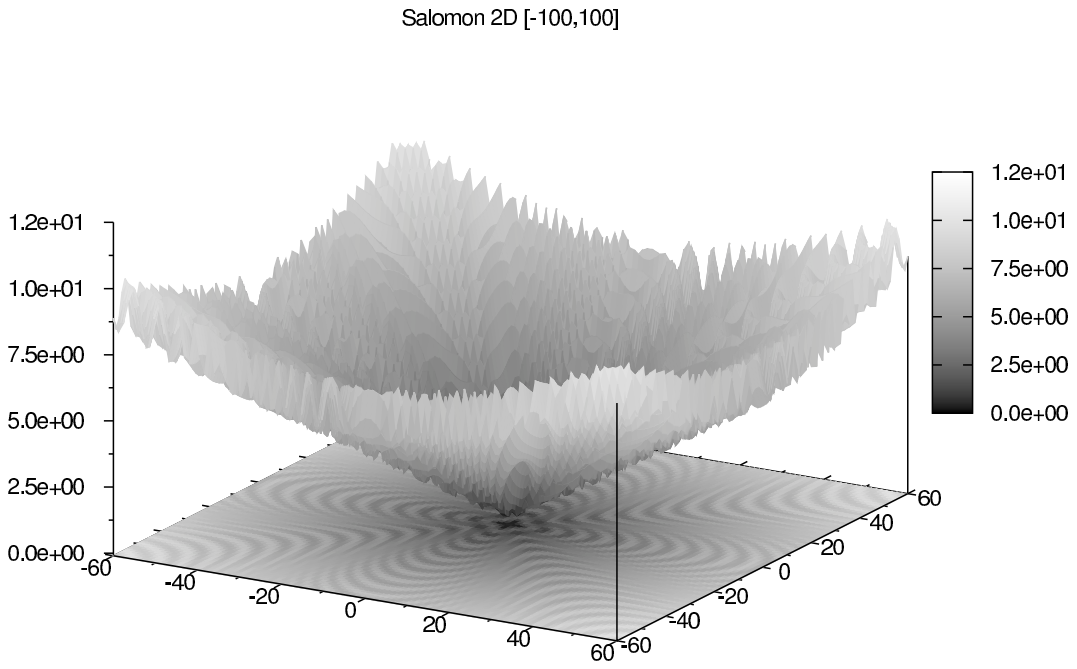


Figura A.9: Representação gráfica da função de Salomon em duas dimensões - vista global.

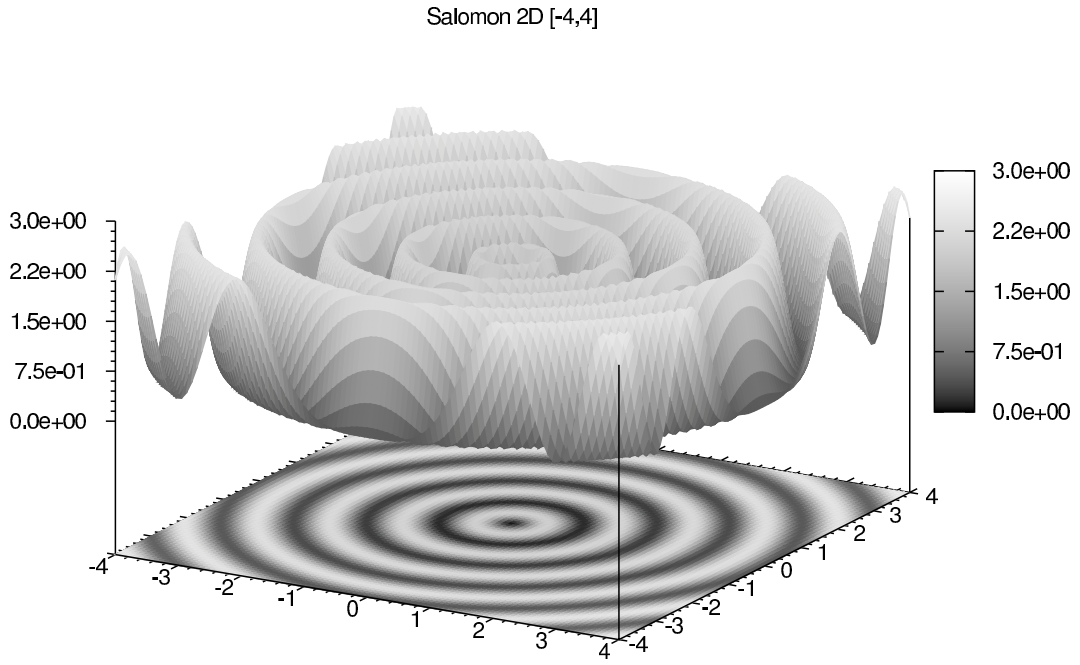


Figura A.10: Representação gráfica da função de Salomon em duas dimensões - detalhe.

A.8 Função de Kursawe

A função de Kursawe é altamente multimodal, com irregularidade no espaçamento dos óptimos, sendo no entanto separável. O espaço de procura é limitado ao hiper-cubo $-1000 \leqslant x_i \leqslant 1000, i = 1, ..., n$, com o óptimo global $f(\mathbf{x}^*) = 0$ para $\mathbf{x}^* = \mathbf{0}$. A figura A.11 apresenta uma representação gráfica desta função a 2 dimensões na totalidade do domínio. A figura A.12 mostra a vizinhança do óptimo global, a qual possui menos simetrias do que nas restantes funções deste grupo.

$$f_8(\mathbf{x}) = \sum_{i=1}^{n} \left(|x_i|^{0.8} + 5 \sin(x_i^3) \right) \tag{A.8}$$

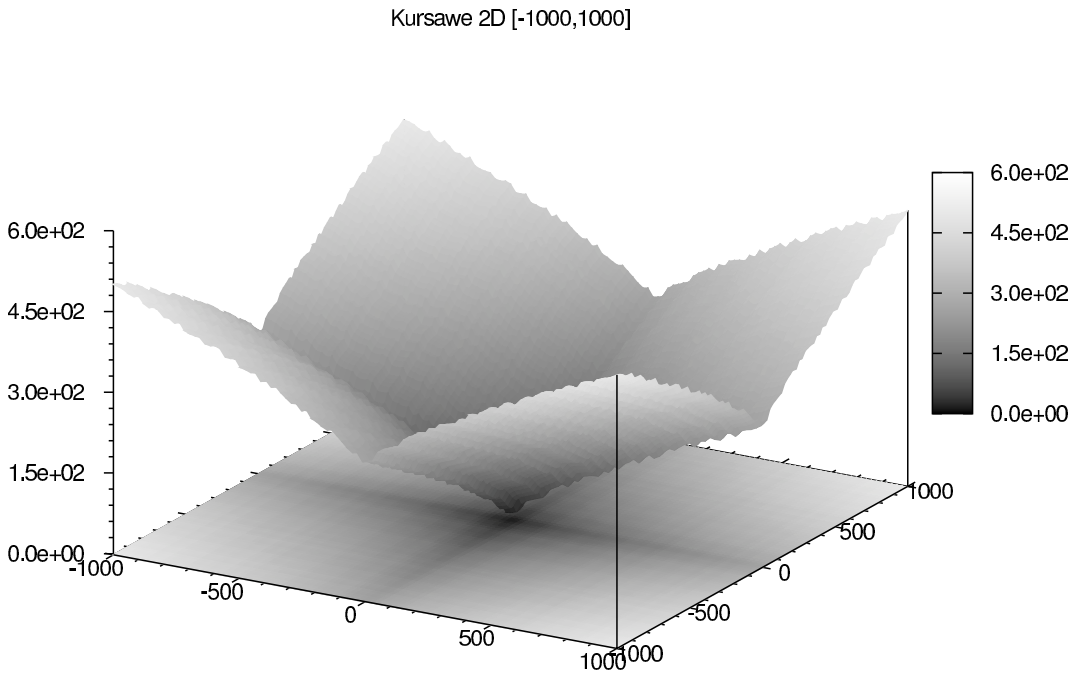


Figura A.11: Representação gráfica da função de Kursawe em duas dimensões - vista global.

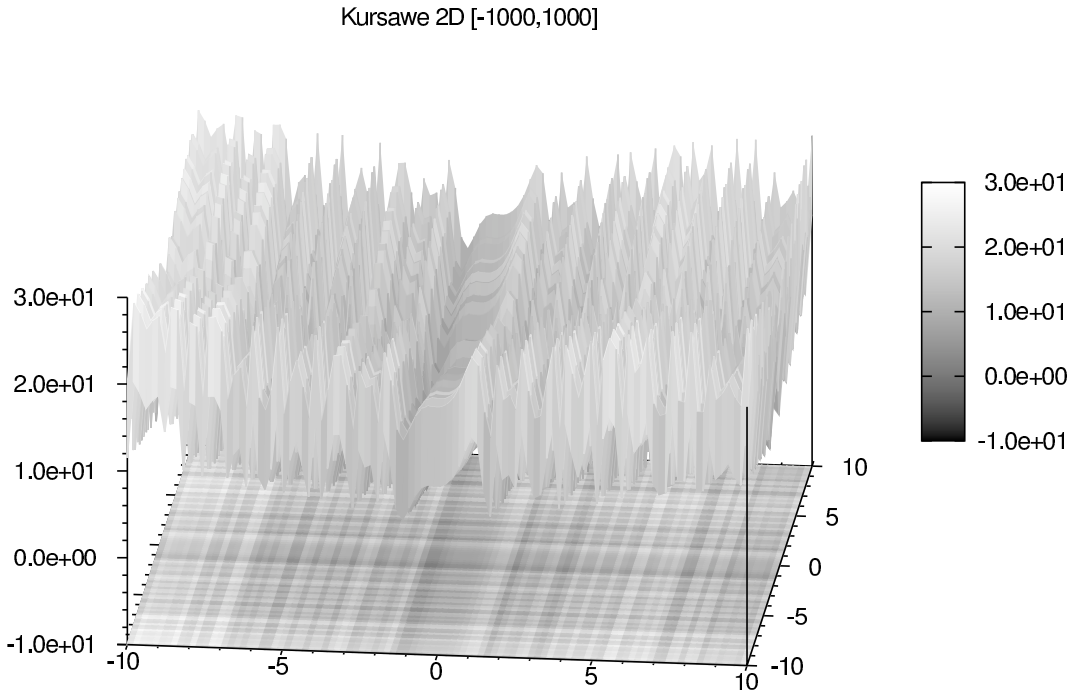


Figura A.12: Representação gráfica da função de Kursawe em duas dimensões - detalhe.

A.9 Função de Shaffer

Esta versão generalizada da função de Shaffer é multimodal e não separável, mantendo alguma regularidade na disposição dos óptimos locais. O espaço de procura é limitado ao hiper-cubo $-100 \leqslant x_i \leqslant 100, i = 1, ..., n$, com o óptimo global $f(\mathbf{x}^*) = 0$ para $\mathbf{x}^* = \mathbf{0}$. A figura A.13 apresenta uma representação gráfica desta função a 2 dimensões na totalidade do domínio.

$$\begin{aligned} f_9(\mathbf{x}) &= \sum_{i=1}^{n-1} s(x_i, x_{i+1}) + s(x_n, x_1), \\ s(x, y) &= 5 + \frac{\sin^2\left(\sqrt{x^2 + y^2}\right) - 0.5}{\left(1 + 0.001(x^2 + y^2)\right)^2} \end{aligned} \tag{A.9}$$

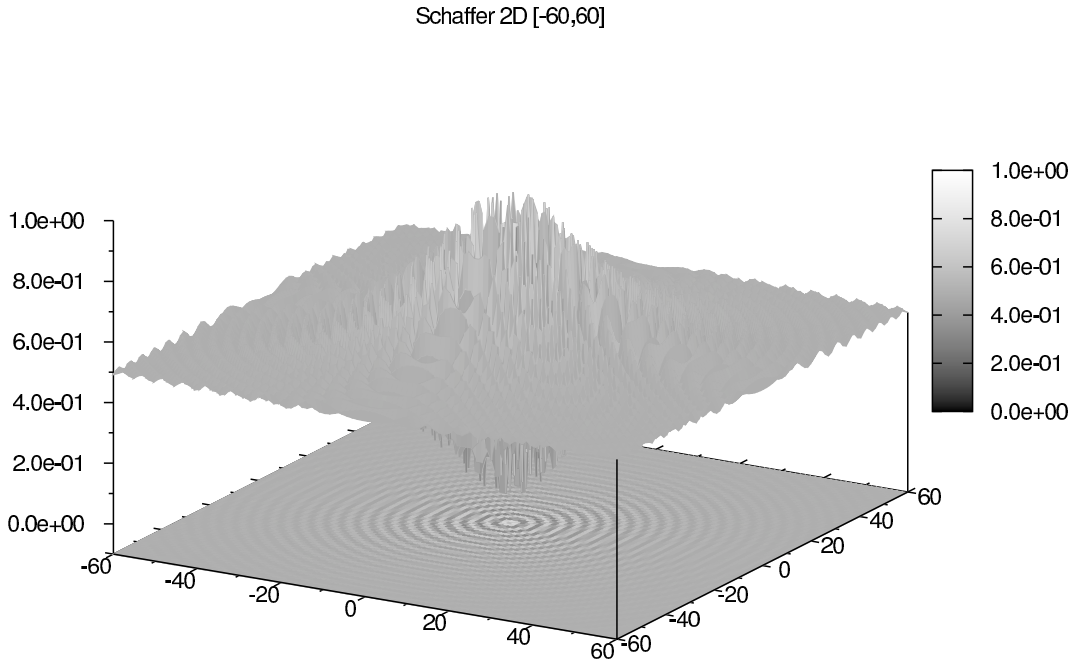


Figura A.13: Representação gráfica da função de Shaffer em duas dimensões.

A.10 Função de Levy-Montalvo

A função de Levy-Montalvo é não separável e apresenta cerca de 15^n óptimos locais, mantendo alguma regularidade na sua disposição. O espaço de procura é limitado ao hiper-cubo $-500 \leqslant x_i \leqslant 500, i = 1, ..., n$, com o óptimo global $f(\mathbf{x}^*) = 0$ para $\mathbf{x}^* = \mathbf{1}$. A figura A.14 apresenta uma representação gráfica desta função a 2 dimensões na totalidade do domínio.

$$f_{10}(\mathbf{x}) = 0.1 \Bigg(\sin^2(3\pi x_1) + \sum_{i=1}^{n-1} (x_i - 1)^2 \left(1 + \sin^2(3\pi x_{i+1})\right) + \\ + (x_n - 1)^2 \left(1 + \sin^2(2\pi x_n)\right) \Bigg) \tag{A.10}$$

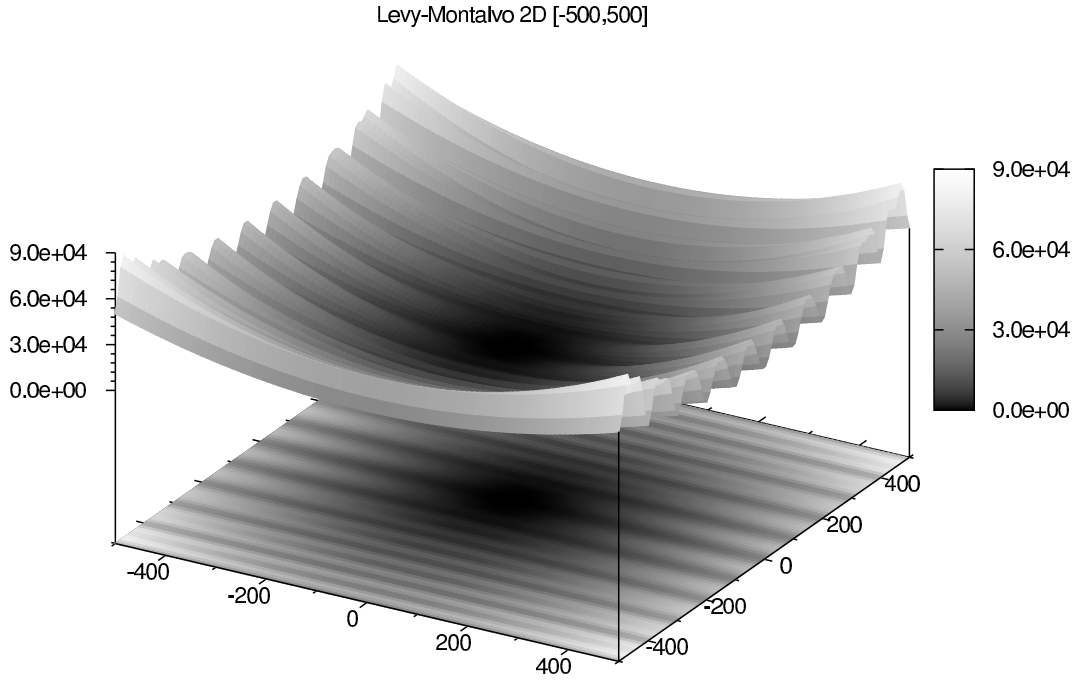


Figura A.14: Representação gráfica da função de Levy-Montalvo em duas dimensões.

A.11 Função de Rosenbrock

A função generalizada de Rosenbrock é multimodal para $n > 3$ e não separável. O maior obstáculo que esta função apresenta à optimização resulta da localização do óptimo local num longo vale plano de forma hiperbólica. Embora o vale em si seja fácil de encontrar pelos algoritmos de optimização, a sua navegação até ao óptimo global, evitando os mínimos locais presentes nos problemas com mais de 3 dimensões, é bastante mais difícil. O espaço de procura é limitado ao hiper-cubo $-5 \leqslant x_i \leqslant 10, i = 1, ..., n$, com o óptimo global $f(\mathbf{x}^*) = 0$ para $\mathbf{x}^* = \mathbf{1}$. A figura A.15 apresenta uma representação gráfica desta função a 2 dimensões, a qual permite observar a região de forma hiperbólica onde se encontra o óptimo.

$$f_{11}(\mathbf{x}) = \sum_{i=1}^{n-1} \left(100 \left(x_i^2 - x_{i+1}\right)^2 + \left(1 - x_i\right)^2\right) \tag{A.11}$$

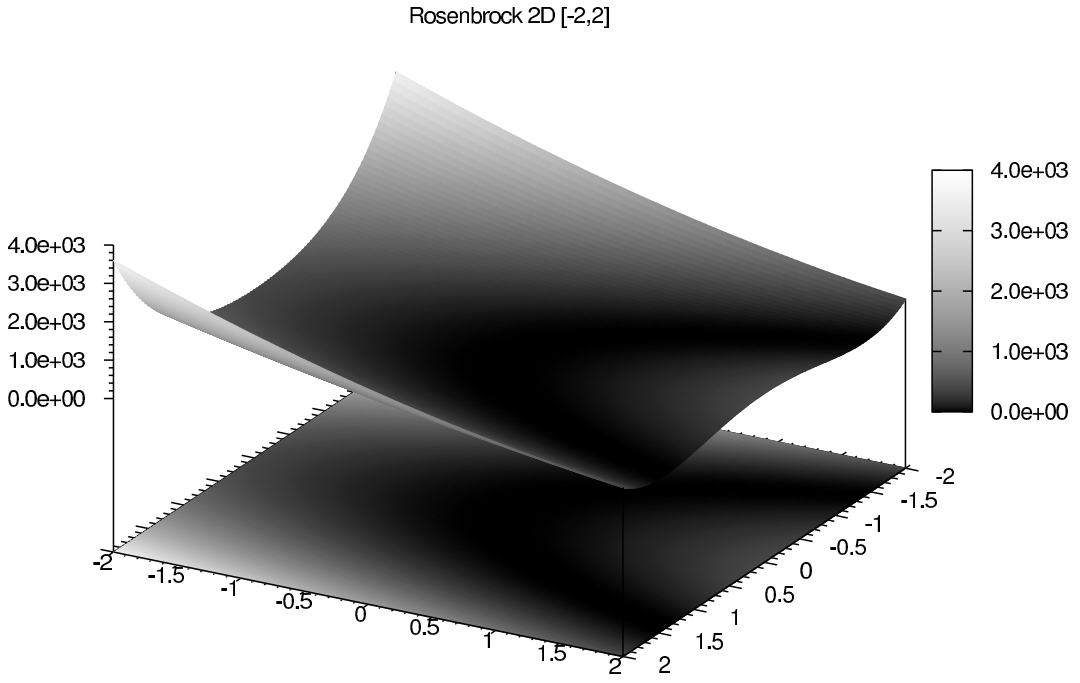


Figura A.15: Representação gráfica da função de Rosenbrock em duas dimensões.

A.12 Função de Michalewicz

A função de Michalewics é multimodal e separável. O óptimo global encontra-se numa indentação profunda no espaço de procura, sendo este quase plano com a excepção de várias destas indentações. O espaço de procura é limitado ao hiper-cubo $0 \leqslant x_i \leqslant \pi, i = 1, ..., n$, com o óptimo global localizado em $x_i^* = \pi, i = 1, ..., n$. A figura A.16 apresenta uma representação gráfica desta função a 2 dimensões, na qual se pode observar as indentações onde o óptimo está alojado.

$$f_{12}(\mathbf{x}) = -\sum_{i=1}^{n} \sin(x_i) \left(\sin\left(\frac{ix_i^2}{\pi} \right) \right)^{20} \tag{A.12}$$

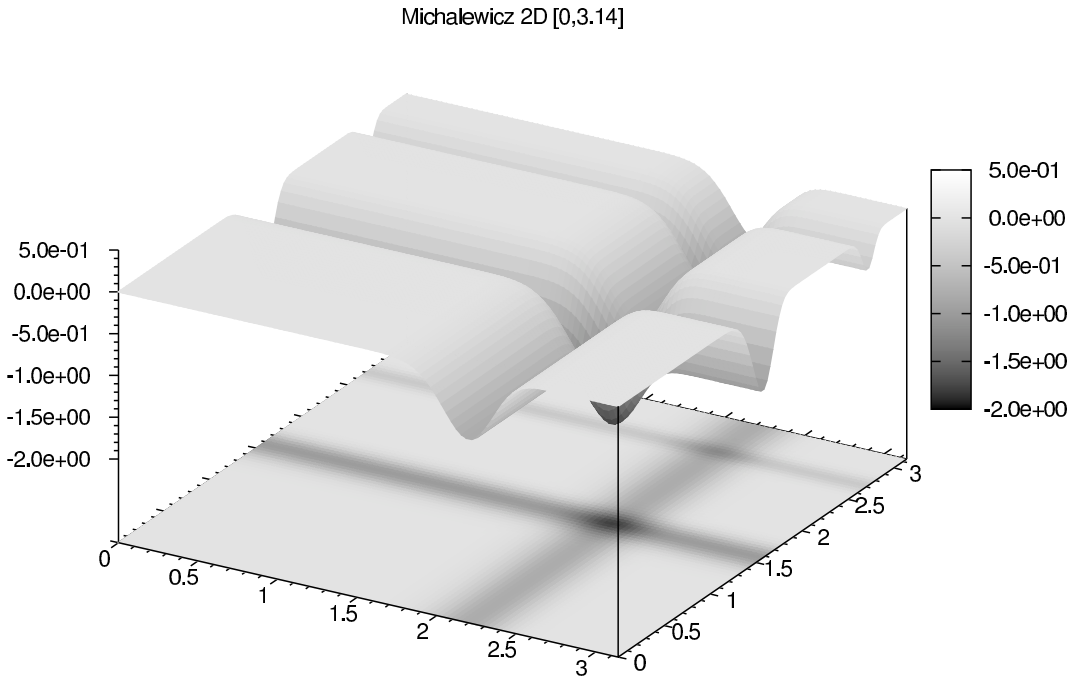


Figura A.16: Representação gráfica da função de Michalewicz em duas dimensões.

A.13 Função de Shubert

A função de Shubert é multimodal e não separável, diferenciando-se das funções anteriores por possuir vários óptimos globais dispostos regularmente no espaço de procura. Estes óptimos globais estão no entanto rodeados por vários óptimos locais. O espaço de procura é limitado ao hiper-cubo $-10 \leqslant x_i \leqslant 10, i = 1, ..., n$. A forma como os óptimos se distribuem no domínio pode ser observada na figura A.17, a qual apresenta uma representação gráfica desta função a 2 dimensões.

$$f_{13}(\mathbf{x}) = \prod_{i=1}^{n} \left(\sum_{j=1}^{5} j \cos\left((j+1)x_i + j\right) \right) \tag{A.13}$$

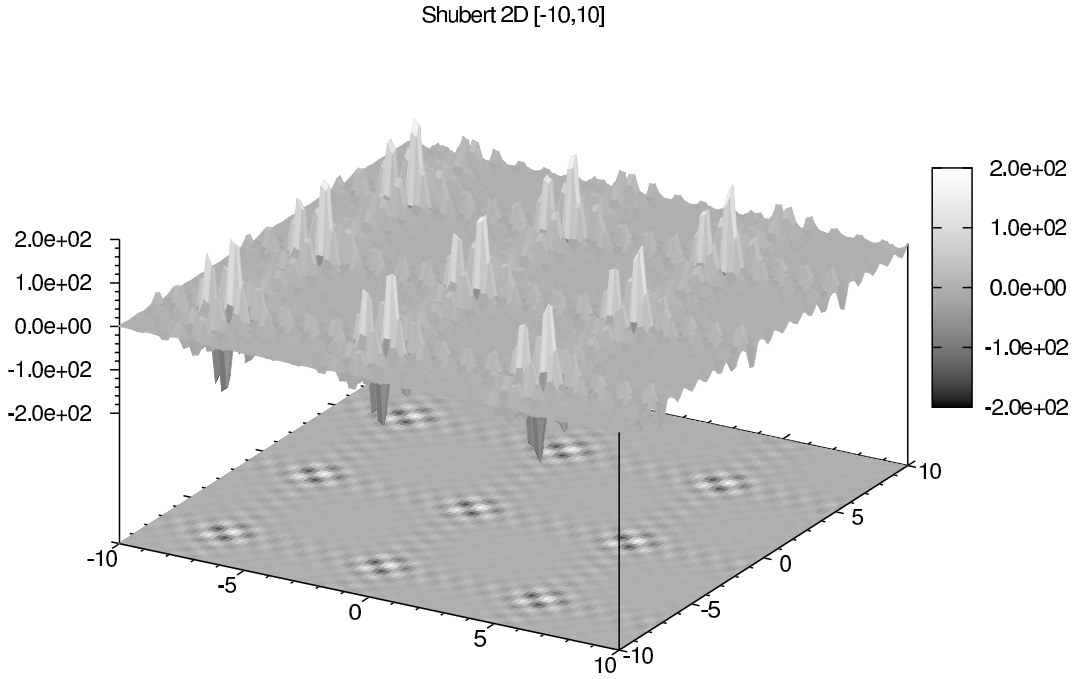


Figura A.17: Representação gráfica da função de Shubert em duas dimensões.

A.14 Função de Schwefel

A função de Schwefel é multimodal e separável, sendo de difícil optimização devido ao óptimo global se encontrar perto do limite do espaço de procura, enquanto que o óptimo local mais próximo se encontra geometricamente distante. Este facto facilita a convergência de muitas execuções dos algoritmos de optimização para esse óptimo local, de onde é muito difícil atingir o máximo global. O espaço de procura é limitado ao hiper-cubo $-500 \leqslant x_i \leqslant 500, i = 1, ..., n$, com o óptimo global $f(\mathbf{x}^*) = 0$ para $x_i^* \approx 420.97, i = 1, ..., n$. O distanciamento entre óptimos e a localização do óptimo local na fronteira do espaço de procura pode ser observada na figura A.18, a qual apresenta uma representação gráfica desta função a 2 dimensões.

$$f_{14}(\mathbf{x}) = 418.98291n + \sum_{i=1}^{n} \left(-x_i^2 \sin(\sqrt{|x_i|})\right) \tag{A.14}$$

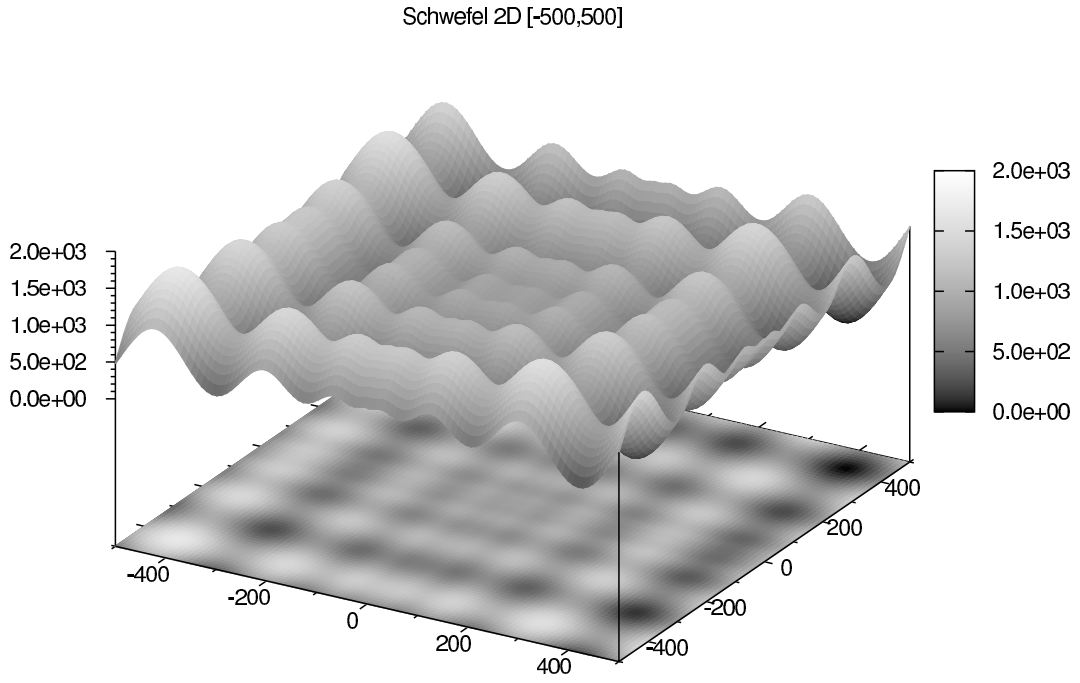


Figura A.18: Representação gráfica da função de Schwefel em duas dimensões.

A.15 Função de Rana

A função de Rana é também bastante difícil de optimizar, já que, além de ser não separável e multimodal, o óptimo local encontra-se numa extremidade do espaço de procura, rodeado de vales com um espaçamento irregular. O espaço de procura é limitado ao hiper-cubo $-520 \leqslant x_i \leqslant 520, i = 1, ..., n$, com o óptimo global $f(\mathbf{x}^*) \approx -512.75$ para $x_i^* \approx 514.04, i = 1, ..., n$. A figura A.19 apresenta uma representação gráfica desta função a 2 dimensões na totalidade do domínio. A figura A.20 mostra a vizinhança do óptimo global, o qual se encontra junto a um dos vértices do espaço de procura.

$$
\begin{aligned}
f_{15}(\mathbf{x}) &= \frac{\sum_{i=1}^{n-1} r(x_i, x_{i+1}) + r(x_n, x_1)}{n}, \\
r(x, y) &= x \sin\left(\sqrt{|y + 1 - x|}\right) \cos\left(\sqrt{|y + 1 + x|}\right) \\
&\quad + (y + 1) \cos\left(\sqrt{|y + 1 - x|}\right) \sin\left(\sqrt{|y + 1 + x|}\right)
\end{aligned}
\tag{A.15}
$$

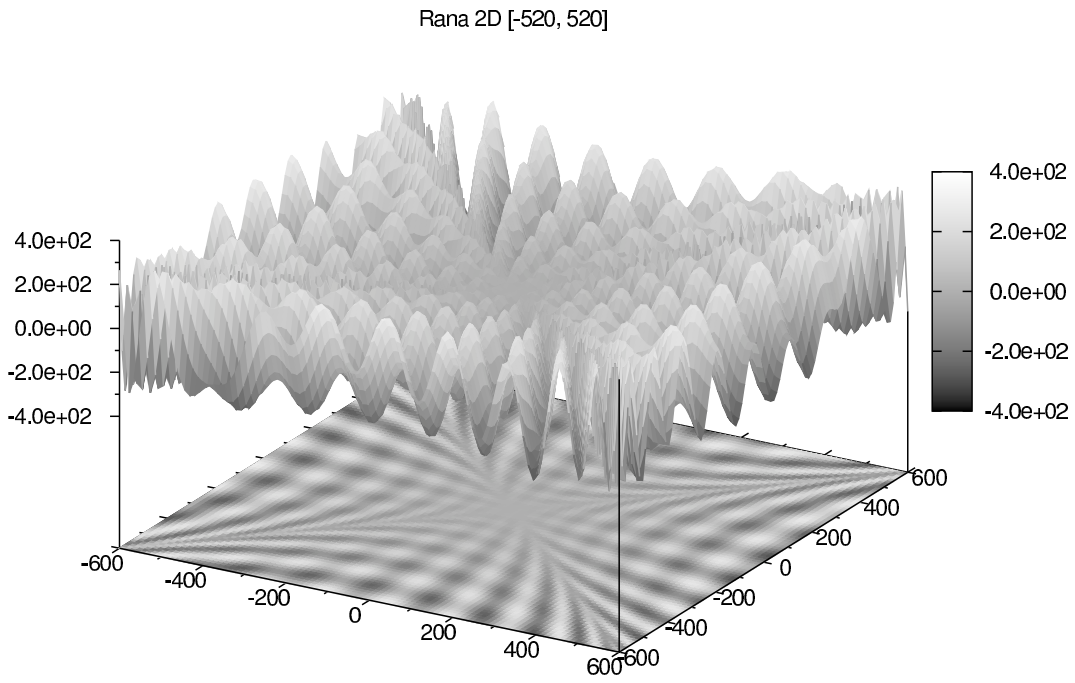


Figura A.19: Representação gráfica da função de Rana em duas dimensões - vista global.

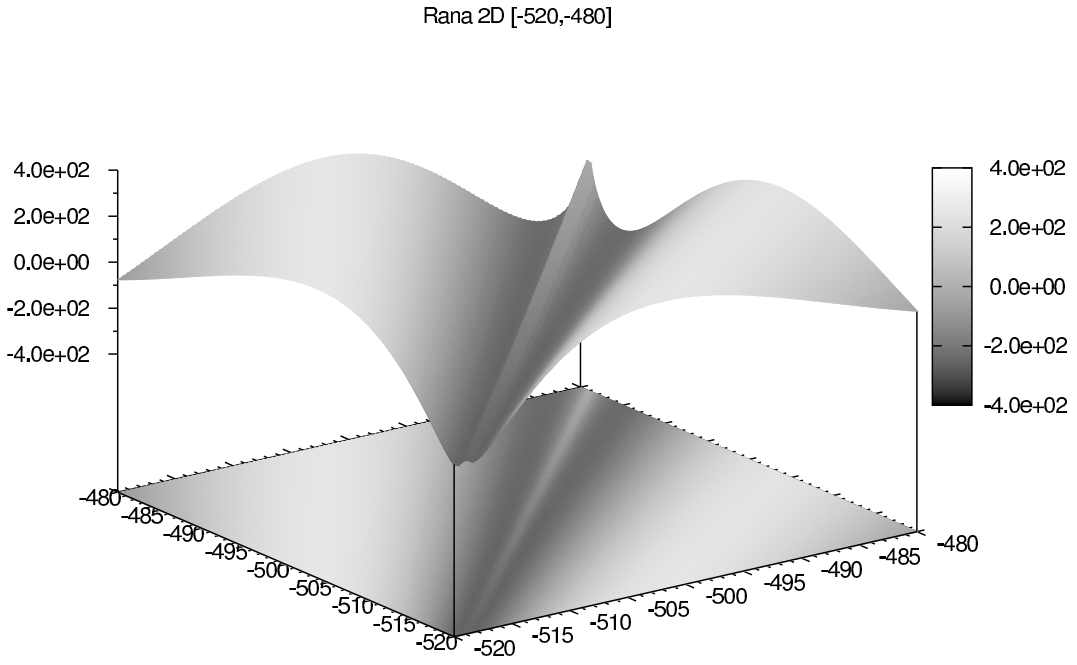


Figura A.20: Representação gráfica da função de Rana em duas dimensões - detalhe.

A.16 Função de Whitley

A função de Whitley é multimodal e não separável, apresentando dois grandes obstáculos à optimização. O primeiro corresponde a uma vasta zona quase plana do espaço de procura que, devido ao gradiente pouco acentuado, facilmente conduz à estagnação dos algoritmos evolucionários. O segundo obstáculo tem a ver com a localização do óptimo local, o qual se encontra cercada por uma pequena zona muito irregular e abundante em óptimos locais. O espaço de procura é limitado ao hiper-cubo $-10 \leqslant x_i \leqslant 10, i = 1, ..., n$, com o óptimo global $f(\mathbf{x}^*) = 0$ para $\mathbf{x}^* = \mathbf{1}$. A figura A.21 apresenta uma representação gráfica desta função a 2 dimensões, a qual permite observar a estrutura quase plana da maior parte do espaço de procura. A figura A.22 mostra a vizinhança bastante ruidosa do óptimo global.

$$f_{16}(\mathbf{x}) = \sum_{i=1}^{n}\sum_{j=1}^{n}\left(\frac{(100(x_i^2 - x_j)^2 + (1 - x_j)^2)^2}{4000} - \right. \\ \left. - \cos\left(100(x_i^2 - x_j)^2 + (1 - x_j)^2\right) + 1\right) \tag{A.16}$$

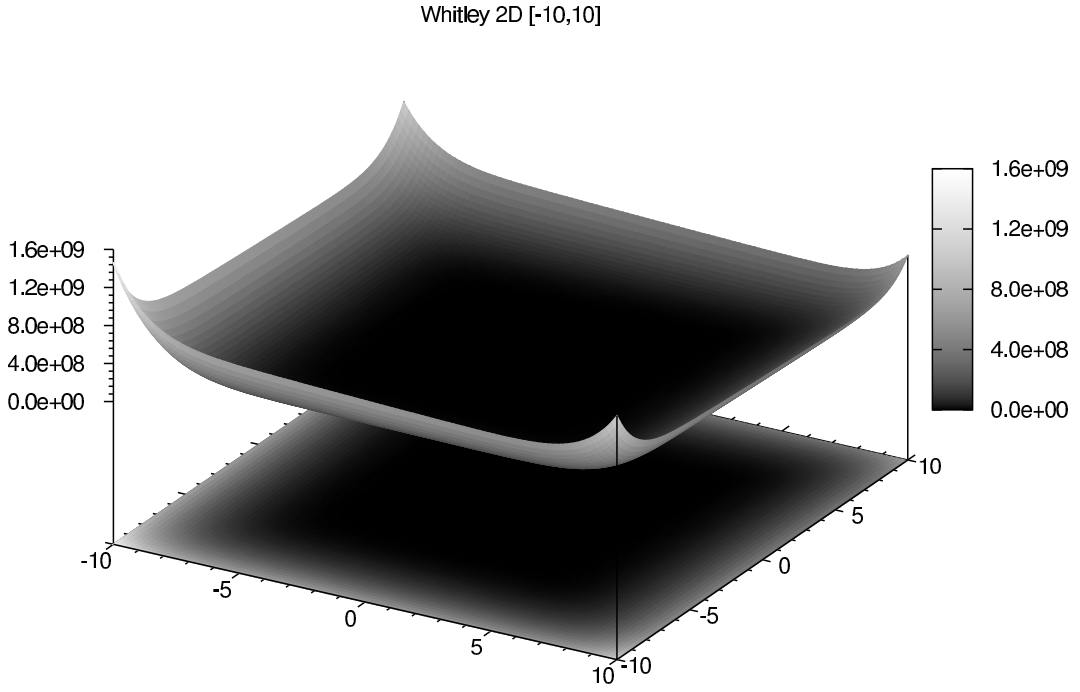


Figura A.21: Representação gráfica da função de Whitley em duas dimensões - vista global.

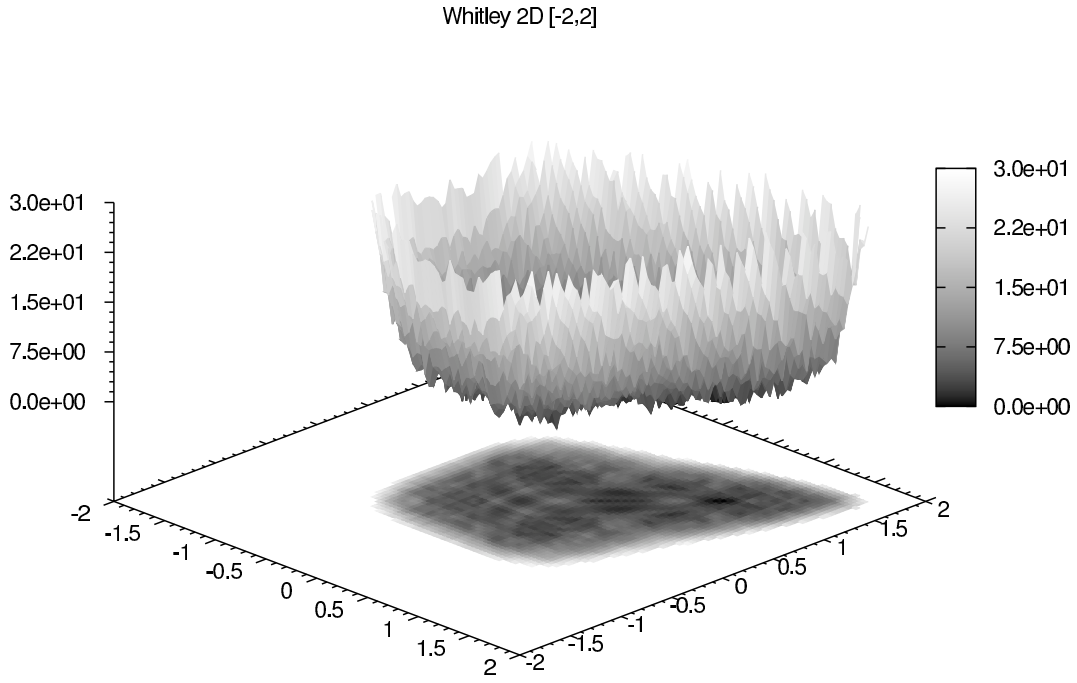


Figura A.22: Representação gráfica da função de Whitley em duas dimensões - detalhe.

Bibliografia

Bibliografia

Ahn H, Lee K, Kim Kj (2006) Global optimization of support vector machines using genetic algorithms for bankruptcy prediction. In: King I, Wang J, Chan L, Wang D (eds) Neural Information Processing, Lecture Notes in Computer Science, vol 4234, Springer Berlin / Heidelberg, pp 420–429

Andrews P (2006) An investigation into mutation operators for particle swarm optimization. In: Evolutionary Computation, 2006. CEC 2006. IEEE Congress on, pp 1044 –1051

Angeline PJ (1998) Evolutionary optimization versus particle swarm optimization: Philosophy and performance differences. In: Proceedings of the 7th International Conference on Evolutionary Programming VII, Springer-Verlag, London, UK, EP '98, pp 601–610

Bache K, Lichman M (2013) UCI machine learning repository

Bahlmann C, Haasdonk B, Burkhardt H (2002) Online handwriting recognition with support vector machines - a kernel approach. In: Proceedings of the Eighth International Workshop on Frontiers in Handwriting Recognition, 2002., pp 49–54

Banks A, Vincent J, Anyakoha C (2007) A review of particle swarm optimization. part i: background and development. Natural Computing 6:467–484

Banks A, Vincent J, Anyakoha C (2008) A review of particle swarm optimization. part ii: hybridisation, combinatorial, multicriteria and constrained optimization, and indicative applications. Natural Computing 7:109–124

Banzhaf W, Nordin P, Keller RE, Francone FD (1997) Genetic Programming : An Introduction : On the Automatic Evolution of Computer Programs and Its Applications (The Morgan Kaufmann Series in Artificial Intelligence). Morgan Kaufmann Publishers

Beyer HG, Schwefel HP (2002) Evolution strategies - a comprehensive introduction. Natural Computing 1:3–52

Bonabeau E, Dorigo M, Theraulaz G (1999) Swarm Intelligence - From Natural to Artificial Systems. Oxford University Press

Brabazon A, Silva A, O'Neill M (2012) Optimal patent design: An agent-based approach. In: Evolutionary Computation (CEC), 2012 IEEE Congress on, IEEE, pp 1–8

Burges CJ (1998) A tutorial on support vector machines for pattern recognition. Data Mining and Knowledge Discovery 2:121–167

Castro LNd, Timmis JI (2003) Artificial immune systems as a novel soft computing paradigm. Soft Computing - A Fusion of Foundations, Methodologies and Applications 7:526–544

Chang CC, Lin CJ (2001) Libsvm: a library for support vector machines

Chen J, Ye J (2008) Training svm with indefinite kernels. In: Proceedings of the 25th international conference on Machine learning, ACM, New York, NY, USA, ICML '08, pp 136–143

Chen Y, Garcia EK, Gupta MR, Rahimi A, Cazzanti L (2009) Similarity-based classification: Concepts and algorithms. The Journal of Machine Learning Research 10:747–776

Clerc M (2006) Particle Swarm Optimization. ISTE

Coore D (2005) Introduction to amorphous computing. In: Banâtre JP, Fradet P, Giavitto JL, Michel O (eds) Unconventional Programming Paradigms, Lecture Notes in Computer Science, vol 3566, Springer Berlin / Heidelberg, pp 97–97

Cristianini N, Scholkopf B (2002) Support vector machines and kernel methods: The new generation of learning machines. Ai Magazine 23(3):31

Cristianini N, Shawe-Taylor J (2000) An Introduction to Support Vector Machines. Cambridge University Press

Darwin C (1872) The Origin of Species by means of Natural Selection; or, the Preservation of Favoured Races in the Struggle for Life, 6th edn

Das S, Suganthan P (2011) Differential evolution: A survey of the state-of-the-art. Evolutionary Computation, IEEE Transactions on 15(1):4 –31

Das S, Abraham A, Konar A (2008) Particle swarm optimization and differential evolution algorithms: Technical analysis, applications and hybridization perspectives. In: Liu Y, Sun A, Loh H, Lu W, Lim EP (eds) Advances of Computational Intelligence in Industrial Systems, Studies in Computational Intelligence, vol 116, Springer Berlin / Heidelberg, pp 1–38

Dawkins R (1990) The Selfish Gene. Oxford University Press

Decoste D, Schölkopf B (2002) Training invariant support vector machines. Machine Learning 46(1-3):161–190

Diosan L, Rogozan A, Pecuchet JP (2007) Evolving kernel functions for svms by genetic programming. In: Machine Learning and Applications, 2007. ICMLA 2007. Sixth International Conference on, pp 19 –24

Dorigo M (1992) Optimization, learning and natural algorithms. PhD Thesis, Politecnico di Milano, Italy

Dorigo M, Blum C (2005) Ant colony optimization theory: A survey. Theoretical Computer Science 344(2-3):243 – 278

Dorigo M, Stützle T (2004) Ant Colony Optimization. Bradford Company, Scituate, MA, USA

Dorigo M, Bonabeau E, Theraulaz G (2000) Ant algorithms and stigmergy. Future Gener Comput Syst 16:851–871

Dréo J, Siarry P (2002) A new ant colony algorithm using the heterarchical concept aimed at optimization of multiminima continuous functions. In: Dorigo M, Di Caro G, Sampels M (eds) Ant Algorithms, Lecture Notes in Computer Science, vol 2463, Springer Berlin / Heidelberg, pp 216–221

Eberhart RC, Kennedy J, Shi Y (2001) Swarm Intelligence. Morgan Kaufmann series in evolutionary computation, Elsevier, Burlington, MA

Engelbrecht A (2010) Heterogeneous particle swarm optimization. In: Dorigo M, Birattari M, Di Caro G, Doursat R, Engelbrecht A, Floreano D, Gambardella L, Groß R, Sahin E, Sayama H, Stützle T (eds) Swarm Intelligence, Lecture Notes in Computer Science, vol 6234, Springer Berlin / Heidelberg, pp 191–202

Engelbrecht AP (2007) Computational Intelligence: An Introduction, 2nd edn. Wiley Publishing

Fan RE, Chen PH, Lin CJ (2005) Working set selection using second order information for training support vector machines. The Journal of Machine Learning Research 6:1889–1918

Fogel DB (1999) Evolutionary Computation: Towards a New Philosophy of Machine Intelligence, 2nd edn. Wiley-IEEE Press

Fogel LJ, Owens AJ, Walsh MJ (1966) Artificial Intelligence through Simulated Evolution. John Wiley, New York, USA

Friedrichs F, Igel C (2005) Evolutionary tuning of multiple svm parameters. Neurocomputing 64:107 – 117, trends in Neurocomputing: 12th European Symposium on Artificial Neural Networks 2004

Frohlich H, Chapelle O, Scholkopf B (2003) Feature selection for support vector machines by means of genetic algorithm. In: Tools with Artificial Intelligence, 2003. Proceedings. 15th IEEE International Conference on, pp 142 – 148

Gagné C, Schoenauer M, Sebag M, Tomassini M (2006) Genetic programming for kernel-based learning with co-evolving subsets selection. In: Runarsson T, Beyer HG, Burke E, Merelo-Guervós J, Whitley L, Yao X (eds) Parallel Problem Solving from Nature - PPSN IX, Lecture Notes in Computer Science, vol 4193, Springer Berlin / Heidelberg, pp 1008–1017

Gao H, Xu W (2011a) A new particle swarm algorithm and its globally convergent modifications. Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on 41(5):1334 –1351

Gao H, Xu W (2011b) Particle swarm algorithm with hybrid mutation strategy. Applied Soft Computing 11(8):5129 – 5142

Garnier S, Gautrais J, Theraulaz G (2007) The biological principles of swarm intelligence. Swarm Intelligence 1:3–31

Gilsberts A, Metta G, Rothkrantz L (2010) Evolutionary optimization of least-squares support vector machines. In: Stahlbock R, Crone SF, Lessmann S (eds) Data Mining, Annals of Information Systems, vol 8, Springer US, pp 277–297

Girdea M, Ciortuz L (2007) A hybrid genetic programming and boosting technique for learning kernel functions from training data. In: Symbolic and Numeric Algorithms for Scientific Computing, 2007. SYNASC. International Symposium on, pp 395 –402

Goldberg DE (1989) Genetic Algorithms in Search, Optimization, and Machine Learning. Addison-Wesley

Grosan C, Abraham A (2007) Hybrid evolutionary algorithms: Methodologies, architectures, and reviews. In: Abraham A, Grosan C, Ishibuchi H (eds) Hybrid Evolutionary Algorithms, Studies in Computational Intelligence, vol 75, Springer Berlin / Heidelberg, pp 1–17

Guo X, Yang J, Wu C, Wang C, Liang Y (2008) A novel ls-svms hyper-parameter selection based on particle swarm optimization. Neurocomputing 71(16-18):3211 – 3215, advances in Neural Information Processing (ICONIP 2006) / Brazilian Symposium on Neural Networks (SBRN 2006)

Haasdonk B (2005) Feature space interpretation of svms with indefinite kernels. IEEE Transactions on Pattern Analysis and Machine Intelligence 27:482–492

Haasdonk B, Bahlmann C (2004) Learning with distance substitution kernels. In: Pattern Recognition, Lecture Notes in Computer Science, vol 3175, Springer Berlin Heidelberg, pp 220–227

Haasdonk B, Keysers D (2002) Tangent distance kernels for support vector machines. In: Proceedings of the 16th International Conference on Pattern Recognition, vol 2, pp 864–868

Hansen N, Kern S (2004) Evaluating the cma evolution strategy on multimodal test functions. In: Yao X, Burke E, Lozano JA, Smith J, Merelo-Guervós JJ, Bullinaria JA, Rowe J, Tino P, Kabán A, Schwefel HP (eds) Parallel Problem Solving from Nature - PPSN VIII, Lecture Notes in Computer Science, vol 3242, Springer Berlin / Heidelberg, pp 282–291

Haupt RL, Haupt SE (2004) Practical Genetic Algorithms with CD-ROM. Wiley-Interscience

Hofmann T, Schölkopf B, Smola A (2008) Kernel methods in machine learning

Holland J (1975) Adaptation in Natural and Artificial Systems. University of Michigan Press, Ann Arbor, MI

Holland J (1992) Genetic Algorithms. Scientific American 267(1)

Hong WC, Chen YF, Chen PW, Yeh YH (2007) Continuous ant colony optimization algorithms in a support vector regression based financial forecasting model. In: Natural Computation, 2007. ICNC 2007. Third International Conference on, vol 1, pp 548 –552

Howley T, Madden M (2005) The genetic kernel support vector machine: Description and evaluation. Artificial Intelligence Review 24:379–395

Huang CL, Dun JF (2008) A distributed pso-svm hybrid system with feature selection and parameter optimization. Applied Soft Computing 8(4):1381 – 1391, soft Computing for Dynamic Data Mining

Huang CL, Wang CJ (2006) A ga-based feature selection and parameters optimizationfor support vector machines. Expert Systems with Applications 31(2):231 – 240

Huang HL, Chang FL (2007) Esvm: Evolutionary support vector machine for automatic feature selection and classification of microarray data. Biosystems 90(2):516 – 528

Huerta E, Duval B, Hao JK (2006) A hybrid ga/svm approach for gene selection and classification of microarray data. In: Rothlauf F, Branke J, Cagnoni S, Costa E, Cotta C, Drechsler R, Lutton E, Machado P, Moore J, Romero J, Smith G, Squillero G, Takagi H (eds) Applications of Evolutionary Computing, Lecture Notes in Computer Science, vol 3907, Springer Berlin / Heidelberg, pp 34–44

Jack AK L Band Nandi (2002) Fault detection using support vector machines and artificial neural networks, augmented by genetic algorithms. Mechanical Systems and Signal Processing 16(2-3):373 – 390

Joachims T (1999) Making Large-Scale SVM Learning Practical, MIT Press, pp 169–184

Jun SH, Oh KW (2006) An evolutionary statistical learning theory. International Journal of Computational Intelligence 3(3):377–384

Kao YT, Zahara E (2008) A hybrid genetic algorithm and particle swarm optimization for multimodal functions. Applied Soft Computing 8(2):849 – 857

Karaboga D, Akay B (2009) A survey: algorithms simulating bee swarm intelligence. Artificial Intelligence Review 31:61–85

Kennedy J, Eberhart R (1995) Particle swarm optimization. In: Neural Networks, 1995. Proceedings., IEEE International Conference on, vol 4, pp 1942 –1948 vol.4

Kennedy J, Eberhart R (1997) A discrete binary version of the particle swarm algorithm. In: Systems, Man, and Cybernetics, 1997. 'Computational Cybernetics and Simulation'., 1997 IEEE International Conference on, vol 5, pp 4104 –4108

Koza J (1994) Genetic programming II: Automatic discovery of reusable programs. Massachusetts Institute of Technology, Cambridge, MA

Koza JR (1992) Genetic programming: on the programming of computers by means of natural selection. MIT Press, Cambridge, MA, USA

Koza JR (2003) Genetic Programming IV: Routine Human-Competitive Machine Intelligence. Kluwer Academic Publishers, Norwell, MA, USA

Langdon W, Poli R (2007) Evolving problems to learn about particle swarm optimizers and other search algorithms. Evolutionary Computation, IEEE Transactions on 11(5):561 –578

Langdon WB, Poli R (2002) Foundations of Genetic Programming. Springer-Verlag

Lee BH, Kim SU, wook Seok J, Won S (2006) Nonlinear system identification based on support vector machine using particle swarm optimization. In: SICE-ICASE, 2006. International Joint Conference, pp 5614 –5618

Li Y, Tong Y, Bai B, Zhang Y (2007) An improved particle swarm optimization for svm training. In: Natural Computation, 2007. ICNC 2007. Third International Conference on, vol 2, pp 611 –615

Lin HT, Lin CJ (2003) A study on sigmoid kernel for svm and the training of non-psd kernels by smo-type methods. Tech. rep., National Taiwan University, Taipei, Department of Computer Science and Information Engineering

Lin SW, Ying KC, Chen SC, Lee ZJ (2008) Particle swarm optimization for parameter determination and feature selection of support vector machines. Expert Systems with Applications 35(4):1817 – 1824

Luss R, d'Aspremont A (2009) Support vector machine classification with indefinite kernels. Mathematical Programming Computation 1:97–118

Melgani F, Bazi Y (2008) Classification of electrocardiogram signals with support vector machines and particle swarm optimization. Information Technology in Biomedicine, IEEE Transactions on 12(5):667 –677

Meyer D, Leisch F, Hornik K (2002) Benchmarking support vector machines. Report Series SFB "Adaptive Information Systems and Modelling in Economics and Management Science" 78, WU Vienna University of Economics and Business, Vienna

Mezura-Montes En, Velázquez-Reyes J, Coello Coello CA (2006) A comparative study of differential evolution variants for global optimization. In: Proceedings of the 8th annual conference on Genetic and evolutionary computation, ACM, New York, NY, USA, GECCO '06, pp 485–492

Mierswa I (2006a) Evolutionary learning with kernels: a generic solution for large margin problems. In: GECCO '06: Proceedings of the 8th annual conference on Genetic and evolutionary computation, ACM, New York, NY, USA, pp 1553–1560

Mierswa I (2006b) Making indefinite kernel learning practical. Tech. rep., Artificial Intelligence Unit, Department of Computer Science, University of Dortmund

Mierswa I (2007) Controlling overfitting with multi-objective support vector machines. In: Proceedings of the 9th annual conference on Genetic and evolutionary computation, ACM, New York, NY, USA, GECCO '07, pp 1830–1837

Mierswa I, Morik K (2008) About the non-convex optimization problem induced by non-positive semidefinite kernel learning. Advances in Data Analysis and Classification 2:241–258

Mierswa I, Wurst M, Klinkenberg R, Scholz M, Euler T (2006) Yale: Rapid prototyping for complex data mining tasks. In: Proceedings of the 12th International Conference on Knowledge Discovery and Data Mining (KDD-06)

Mitchell M (1996) An introduction to genetic algorithms. MIT Press, Cambridge, MA, USA

Monson C, Seppi K (2005) Linear equality constraints and homomorphous mappings in pso. In: Evolutionary Computation, 2005. The 2005 IEEE Congress on, vol 1, pp 73 –80

Moreno PJ, Ho PP, Vasconcelos N (2003) A kullback-leibler divergence based kernel for svm classification in multimedia applications. In: Advances in neural information processing systems

Moscato P (1989) On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms. Tech. Rep. report 826, Caltech Concurrent Computation Program

Moscato P, Cotta C (2003) A gentle introduction to memetic algorithms. In: Hillier FS, Glover F, Kochenberger G (eds) Handbook of Metaheuristics, International Series in Operations Research and Management Science, vol 57, Springer New York, pp 105–144

Nguyen HN, Ohn SY, Choi WJ (2004) Combined kernel function for support vector machine and learning method based on evolutionary algorithm. In: Pal NR, Kasabov N, Mudi RK, Pal S, Parui SK (eds) Neural Information Processing, Lecture Notes in Computer Science, vol 3316, Springer Berlin / Heidelberg, pp 1273–1278

Montes de Oca M, Pena J, Stutzle T, Pinciroli C, Dorigo M (2009) Heterogeneous particle swarm optimizers. In: Evolutionary Computation, 2009. CEC '09. IEEE Congress on, pp 698 –705

Ong CS, Mary X, Canu S, Smola AJ (2004) Learning with non-positive kernels. In: Proceedings of the twenty-first international conference on Machine learning, ACM, New York, NY, USA, ICML '04, pp 81–

Ong YS, Lim MH, Zhu N, Wong KW (2006) Classification of adaptive memetic algorithms: a comparative study. Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on 36(1):141 –152

Osuna E, Freund R, Girosi F (1997a) Support vector machines: Training and applications. Tech. Rep. AIM-1602

Osuna E, Freund R, Girosi F (1997b) Training support vector machines: an application to face detection. Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition 6:130–136

Pai PF (2006) System reliability forecasting by support vector machines with genetic algorithms. Mathematical and Computer Modelling 43(3-4):262 – 274

Paquet U, Engelbrecht A (2003a) A new particle swarm optimiser for linearly constrained optimisation. In: Evolutionary Computation, 2003. CEC '03. The 2003 Congress on, vol 1, pp 227 – 233 Vol.1

Paquet U, Engelbrecht A (2003b) Training support vector machines with particle swarms. In: Neural Networks, 2003. Proceedings of the International Joint Conference on, vol 2, pp 1593 – 1598

Paquet U, Engelbrecht AP (2007) Particle swarms for linearly constrained optimisation. Fundam Inf 76:147–170

Pedersen M (2010a) Good parameters for differential evolution. Technical Report HL 1002, Hvass Laboratories

Pedersen M (2010b) Good parameters for particle swarm optimization. Technical Report HL 1001, Hvass Laboratories

Penev K, Littlefair G (2005) Free search - a comparative analysis. Information Sciences 172(1-2):173 – 193

Peng S, Xu Q, Ling XB, Peng X, Du W, Chen L (2003) Molecular classification of cancer types from microarray data using the combination of genetic algorithms and support vector machines. FEBS Letters 555(2):358 – 362

Petalas Y, Parsopoulos K, Vrahatis M (2007) Memetic particle swarm optimization. Annals of Operations Research 156:99–127

Phienthrakul T, Kijsirikul B (2005) Evolutionary strategies for multi-scale radial basis function kernels in support vector machines. In: Proceedings of the 2005 conference on Genetic and evolutionary computation, ACM, New York, NY, USA, GECCO '05, pp 905–911

Platt JC (1999) Fast training of support vector machines using sequential minimal optimization, MIT Press, Cambridge, MA, USA, pp 185–208

Poli R (2003) A simple but theoretically-motivated method to control bloat in genetic programming. In: Ryan C, Soule T, Keijzer M, Tsang E, Poli R, Costa E (eds) Genetic Programming, Lecture Notes in Computer Science, vol 2610, Springer Berlin / Heidelberg, pp 43–76

Poli R, Kennedy J, Blackwell T (2007) Particle swarm optimization. Swarm Intelligence 1:33–57

Poli R, Langdon WB, McPhee NF (2008) A field guide to genetic programming. Published via http://lulu.com

Price KV, Storn RM, Lampinen JA (2005) Differential Evolution A Practical Approach to Global Optimization, Springer-Verlag, Berlin, Germany. Natural Computing Series

Rahnamayan S, Tizhoosh H, Salama M (2008) Opposition-based differential evolution. Evolutionary Computation, IEEE Transactions on 12(1):64 –79

Raymer M, Punch W, Goodman E, Kuhn L, Jain A (2000) Dimensionality reduction using genetic algorithms. Evolutionary Computation, IEEE Transactions on 4(2):164 –171

Rechenberg I (1973) Evolutionsstrategie – Optimierung technisher Systeme nach Prinzipien der biologischen Evolution. Frommann-Holzboog, Stuttgart, GER

Rossi A, de Carvalho A (2008) Bio-inspired optimization techniques for svm parameter tuning. In: Neural Networks, 2008. SBRN '08. 10th Brazilian Symposium on, pp 57 –62

Runarsson T, Sigurdsson S (2004) Asynchronous parallel evolutionary model selection for support vector machines. Neural Information Processing – Letters and Reviews 3:59–67

Rüping S (2000) mySVM-Manual. University of Dortmund, Lehrstuhl Informatik 8

Samanta B, Nataraj C (2009) Application of particle swarm optimization and proximal support vector machines for fault detection. Swarm Intelligence 3:303–325

Schwefel HP (1974) Numerische Optimierung von Computer-Modellen. Birkhäuser

Schwefel HP (1981) Numerical optimization of computer models. Wiley & Sons, Chichester

Shawe-Taylor J, Cristianini N (2004) Kernel methods for pattern analysis. Cambridge Univ. Press, Cambridge

Shelokar P, Siarry P, Jayaraman V, Kulkarni B (2007) Particle swarm and ant colony algorithms hybridized for improved continuous optimization. Applied Mathematics and Computation 188(1):129 – 142

Shen Q, Shi WM, Kong W, Ye BX (2007) A combination of modified particle swarm optimization algorithm and support vector machine for gene selection and tumor classification. Talanta 71(4):1679 – 1683

Shi Y, Eberhart R (1999) Empirical study of particle swarm optimization. In: Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on, vol 3, pp (xxxvii+2348)

Silva A, Gonçalves T (2013a) Training support vector machines with an heterogeneous particle swarm optimizer. In: Adaptive and Natural Computing Algorithms, 11th International Conference, ICANNGA 2013, Lecture Notes in Computer Science, vol 7824, Springer Berlin Heidelberg, pp 100–109

Silva A, Gonçalves T (2013b) Using a scouting predator-prey optimizer to train support vector machines with non psd kernels. In: Nature Inspired Cooperative Strategies for Optimization, NICSO 2011, Studies in Computational Intelligence, Springer Berlin Heidelberg

Silva A, Gonçalves T (2013c) Using scout particles to improve a predator-prey optimizer. In: Adaptive and Natural Computing Algorithms, 11th International Conference, ICANNGA 2013, Lecture Notes in Computer Science, vol 7824, Springer Berlin Heidelberg, pp 130–139

Silva A, Neves A, Costa E (2002a) Chasing the swarm: A predator prey approach to function optimisation. In: Proceedings of the MENDEL2002, 8th International Conference on Soft Computing. Brno, Czech Republic, pp 103–110

Silva A, Neves A, Costa E (2002b) An empirical comparison of particle swarm and predator prey optimisation. In: O'Neill M, Sutcliffe R, Ryan C, Eaton M, Griffith N (eds) Artificial Intelligence and Cognitive Science, 13th Irish International Conference, AICS 2002, Lecture Notes in Computer Science, vol 2464, Springer Berlin Heidelberg, pp 103–110

Silva A, Neves A, Costa E (2003) Sappo: A simple, adaptable, predator prey optimiser. In: Progress in Artificial Intelligence, 11th Portuguese Conference on Artificial Intelligence, EPIA 2003, Lecture Notes in Computer Science, vol 2902, Springer Berlin Heidelberg, pp 59–73

Silva A, Neves A, Gonçalves T (2012a) An heterogeneous particle swarm optimizer with predator and scout particles. In: Autonomous and Intelligent Systems, Third

International Conference, AIS 2012, Lecture Notes in Computer Science, vol 7326, Springer, pp 200–208

Silva AP, Silva A, Rodrigues IP (2012b) Biopos: Biologically inspired algorithms for pos tagging,. In: Proceedings of the 1st Workshop on Optimization Techniques for Language Technology, OPTHLT 2012 / COLING 2012, pp 1–16

Silva AP, Silva A, Rodrigues I (2013) Pso-tagger: A new biologically inspired approach to the part-of-speech tagging problem. In: Adaptive and Natural Computing Algorithms, 11th International Conference, ICANNGA 2013, Springer Berlin Heidelberg, pp 90–99

Smith J (2007) Coevolving memetic algorithms: A review and progress report. Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on 37(1):6 –17, DOI 10.1109/TSMCB.2006.883273

Socha K, Dorigo M (2008) Ant colony optimization for continuous domains. European Journal of Operational Research 185(3):1155 – 1173

de Souza B, de Carvalho A, Calvo R, Ishii R (2006) Multiclass svm model selection using particle swarm optimization. In: Hybrid Intelligent Systems, 2006. HIS '06. Sixth International Conference on, pp 31 –31

Statnikov A, Aliferis CF, Tsamardinos I, Hardin D, Levy S (2005) A comprehensive evaluation of multicategory classification methods for microarray gene expression cancer diagnosis. Bioinformatics 21:631–643

Stoean R, Preuss M, Stoean C, Dumitrescu D (2007) Concerning the potential of evolutionary support vector machines. In: Evolutionary Computation, 2007. CEC 2007. IEEE Congress on, pp 1436 –1443

Stoean R, Stoean C, Lupsor M, Stefanescu H, Badea R (2011) Evolutionary-driven support vector machines for determining the degree of liver fibrosis in chronic hepatitis c. Artificial Intelligence in Medicine 51(1):53 – 65

Storn R, Price K (1997) Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces. Journal of Global Optimization 11:341–359

Sullivan KM, Luke S (2007) Evolving kernels for support vector machine classification. In: Proceedings of the 9th annual conference on Genetic and evolutionary computation, ACM, New York, NY, USA, GECCO '07, pp 1702–1707

Tipping ME (2001) Sparse bayesian learning and the relevance vector machine. J Mach Learn Res 1:211–244

Tizhoosh H (2005) Opposition-based learning: A new scheme for machine intelligence. In: Computational Intelligence for Modelling, Control and Automation, 2005 and International Conference on Intelligent Agents, Web Technologies and Internet Commerce, International Conference on, vol 1, pp 695 –701

Ventresca M, Tizhoosh H (2006) Improving the convergence of backpropagation by opposite transfer functions. In: Neural Networks, 2006. IJCNN '06. International Joint Conference on, pp 4777–4784

Vesterstrom J, Thomsen R (2004) A comparative study of differential evolution, particle swarm optimization, and evolutionary algorithms on numerical benchmark problems. In: Evolutionary Computation, 2004. CEC2004. Congress on, vol 2, pp 1980–1987

Vlachos P (2013) Statlib datasets archive

Wang H, Li H, Liu Y, Li C, Zeng S (2007) Opposition-based particle swarm algorithm with cauchy mutation. In: Evolutionary Computation, 2007. CEC 2007. IEEE Congress on, pp 4750 –4756

Wang X, Zhang H, Zhang C, Cai X, Wang J, Ye M (2006) Time series prediction using ls-svm with particle swarm optimization. In: Wang J, Yi Z, Zurada J, Lu BL, Yin H (eds) Advances in Neural Networks - ISNN 2006, Lecture Notes in Computer Science, vol 3972, Springer Berlin / Heidelberg, pp 747–752

Weise T (2009) Global optimization algorithms – theory and application. Self-published, available at `www.it-weise.de`

Wu CH, Tzeng GH, Goo YJ, Fang WC (2007) A real-valued genetic algorithm to optimize the parameters of support vector machine for predicting bankruptcy. Expert Systems with Applications 32(2):397 – 408

Wu G, Chang EY, Zhang Z (2005) An analysis of transformation on non-positive semidefinite similarity matrix for kernel machines. In: Proceedings of the 22nd International Conference on Machine Learning

Yang J, Honavar V (1998) Feature subset selection using a genetic algorithm. Intelligent Systems and their Applications, IEEE 13(2):44 –49

Yuan SF, Chu FL (2007) Fault diagnostics based on particle swarm optimisation and support vector machines. Mechanical Systems and Signal Processing 21(4):1787 – 1798

Contactos:
Universidade de Évora
Instituto de Investigação e Formação Avançada - IIFA
Palácio do Vimioso | Largo Marquês de Marialva, Apart. 94
7002-554 Évora | Portugal
Tel: (+351) 266 706 581
Fax: (+351) 266 744 677
email: iifa@uevora.pt