7

METODOLOGIA GeRL NA OPTIMIZAÇÃO DE PROBLEMAS DE *LAYOUTS* DE INSTALAÇÕES

Foi já demonstrado que o espaço de soluções resultante do modelo proposto para solucionar PPLI com o recurso a meta-interpretadores da PLR(*DF*) é demasiado vasto para que possa ser completamente explorado em tempo útil. Foi ainda concluído no capítulo 5 que, um método para encontrar em tempo útil boas soluções para este tipo de problemas, deveria efectuar uma amostragem uniforme de todo o espaço de soluções. Um conjunto de métodos, denominados por computação evolucionária, têm sido aplicados com sucesso na resolução de problemas complexos como é o caso dos PPLI. Como se referiu no capítulo 6, estes métodos são métodos estocásticos que aliam as características da pesquisa local com a pesquisa global para explorar as regiões mais prometedoras do espaço de soluções

Este capítulo tem por objectivo fundamental a avaliação da metodologia *GeRL*

na resolução de PPLI, sendo dado um especial ênfase aos operadores genéticos que foram desenvolvidos. Para o efeito foi desenvolvido um protótipo, a que se deu a designação de *LayGeRL* (geração de ***Lay****out* de instalações usando algoritmos **Ge**néticos com a tecnologia das **R**estrições e da **L**ógica).

7.1 A Computação Evolucionária na Resolução de Problemas de *Layout*

Os métodos de computação evolucionária, e fundamentalmente os Algoritmos Genéticos (AG), têm vindo a ser largamente utilizados em áreas do conhecimento tão diversas como, por exemplo, a optimização combinatória, a optimização numérica de funções, o processamento de imagem, problemas de projecto e a aprendizagem. Em geral estes métodos têm mostrado ser bastante bem sucedidos na resolução de problemas complexos. Como se tem vindo a demonstrar ao longo da tese os PPLI, em particular os relacionados com os sistemas produtivos, pertencem a esta categoria de problemas. Em boa verdade, grande parte dos problemas de optimização de produção são problemas complexos e por isso candidatos a serem solucionados por métodos de computação evolucionária.

Dimopoulos e Zalzala (2000) publicaram um trabalho onde passam em revista os mais recentes desenvolvimentos no campo da computação evolucionária na resolução dos principais problemas de optimização da produção. Entre os diferentes problemas de optimização de produção destacam-se os problemas de escalonamento *Job-Shop*[1] e *Flow-Shop* , de planeamento de processos, de optimização de sistemas de produção celulares, de optimização de linhas de produção e de optimização de desenho ou projecto. No âmbito do problema tratado na tese, têm um particular interesse os problemas de optimização de sistemas de produção celulares dado que estes se inserem na categoria dos PPLI. O projecto de sistemas de produção celulares compreende três fases: i) o agrupamento de máquinas em diferentes células; ii) a

[1] Uma tradução possível para este termos é “oficina de tarefas”.

[2] Pode ser traduzido como “oficina de fluxos”.

geração do *layout* das células nas plantas fabris; e iii) a geração do *layout* das máquinas dentro de cada célula. As duas últimas fases são de facto casos particulares de PPLI. Numa das fases as células são tratadas como instalações e na outra cada célula é um PPLI em que a área da célula é a planta e as máquinas são as instalações.

Diversas abordagens têm sido propostas para solucionar PPLI com o recurso a métodos da computação evolucionária, tanto em termos do modelo adoptado como relativamente à representação escolhida. Segundo Dimopoulos e Zalzala, as primeiras abordagens para solucionar PPLI utilizando métodos de computação evolucionária foram propostas por Cohoon *et al.*(1992) e Tam (1992a). Em ambos os casos, as soluções para os problemas são representadas por árvores de corte que foram já descritas na secção 2.5.2.

Uma abordagem diferente, que recorre à computação evolucionária, foi adoptada por Tate e Smith (1995) com o FLEX-BAY. Esta abordagem parte do formalismo do QAP de modo a usar uma estrutura para o *layout* baseado em bandas flexíveis[3] que acomodam instalações de tamanhos diferentes. A planta fabril é dividida num determinado número de bandas de uma extremidade à outra e numa determinada direcção. Estas bandas são então dividas em partes que acomodam as instalações por intermédio de cortes transversais (ver Figura 7-1). A representação das soluções baseia-se numa representação por permutações que determina não só a banda atribuída a cada instalação bem como a divisão das bandas.

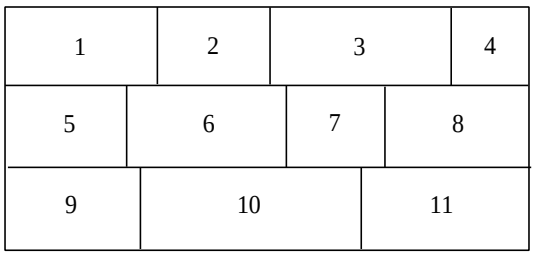


Figura 7-1: Um exemplo de um *layout* baseado em bandas flexíveis para 11 instalações.

Para solucionar problemas de *layout*, considerando instalações de áreas diferentes, pela utilização de AG, Kochhar *et al.* (1996) desenvolveram um método

[3] Termo original em inglês é *flexible-bay layout structure*.

que passa pela divisão de cada instalação em blocos quadrados. Cada bloco é tratado como uma unidade de área. O número de blocos atribuídos a cada instalação depende da área que esta ocupa.. A representação das soluções é efectuada por intermédio de uma cadeia que contém o número de cada instalação segundo uma sequência da esquerda para a direita e de cima para baixo. O *layout* indicado na Figura 7-2 é representado pela cadeia {1, 1, 6, 1, 6, 6, 5, 2, 2, 3, 4, 4}. Este método foi mais tarde estendido de forma a resolver problemas de *layout* considerando que as instalações podem ser dispostas por vários pisos (Kochhar, 1998).

1	1	6
1	6	6
5	2	2
3	4	4

Figura 7-2: *Layout* de instalações de áreas diferentes representado por quadrados unitários.

Uma outra abordagem interessante foi proposta por Banerjee *et al.* (1997) e segue um modelo do problema baseado no formalismo da programação inteira mista. A representação das soluções escolhida é baseada em grafos onde cada nó representam as instalações e os ramos correspondem ao fluxo de material entre as instalações. São usados AG como uma parte do algoritmo global, procurando-se transformar o problema numa série iterativa de problemas de programação linear.

7.2 A Metodologia *GeRL* na Resolução de PPLI

Nesta secção é descrita uma abordagem desenvolvida baseada na metodologia *GeRL* para solucionar PPLI (Tavares *et al.*, 2000a; Tavares *et al.*, 2000b). Recorde-se que a metodologia *GeRL*, descrita na secção 6.4.2, passa essencialmente pela utilização de um AG na optimização de problemas com aplicações desenvolvidas segundo o paradigma da Programação Lógica por Restrições com Domínios Finitos (PLR(*DF*)). O protótipo *LayGeRL* desenvolvido, que implementa esta abordagem, consiste, de uma forma simplificada, numa versão modificada do *LaRLo* onde o algoritmo *Branch* & *Bound* (*B&B*) é substituído por um AG cujos os operadores genéticos são implementados segundo o paradigma da PLR(*DF*), tal como define a metodologia *GeRL*.

7.2.1 Arquitectura do *LayGeRL*

Como o *LayGeRL* é uma versão modificada do *LaRLo*, este partilha com ele a mesma sequência de etapas. Relembrando, estas correspondem à criação das variáveis de decisão com os seus respectivos domínios, à colocação das restrições do problema, à definição da função de custo e finalmente a etapa responsável pelo procedimento de optimização. É precisamente nesta última etapa que residem as principais diferenças relativamente ao *LaRLo*. Todas as restantes etapas são idênticas, mesmo as etapas preliminares relacionadas com a recolha da informação e com o cálculo do valor de fluxo entre as diferentes instalações. Como é referido na secção 5.3, o procedimento de optimização do *LaRLo* envolve a utilização de um algoritmo *B&B*, possivelmente embebido no meta-interpretador de PLR(*DF*), com um procedimento de etiquetagem das variáveis de decisão adequado. No caso do *LayGeRL*, o procedimento de optimização é um AG que utiliza operadores genéticos implementados segundo o paradigma da PLR (ver Figura 7-3). Em termos práticos, estes operadores genéticos não são mais que procedimentos de etiquetagem específicos que exploram apenas uma reduzida região de todo o espaço de soluções.

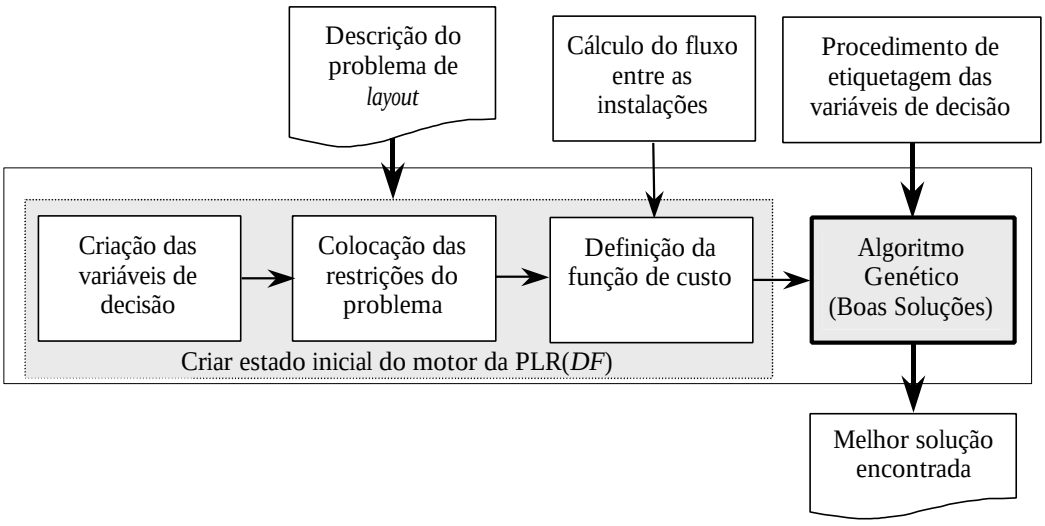


Figura 7-3: Sequência das etapas principais do *LayGeRL*.

Tal como foi referido na secção 6.4.2 qualquer aplicação que segue a metodologia *GeRL*, como é o caso do *LayGeRL*, divide-se em duas partes: o processo principal responsável pelas tarefas de optimização e o motor da PLR que

realiza determinadas tarefas de que são exemplo os operadores genéticos (Figura 7-4). De referir que as três primeiras etapas da sequência de etapas do *LayGeRL* não são mais que procedimento de criação do estado inicial (Π) do motor da PLR.

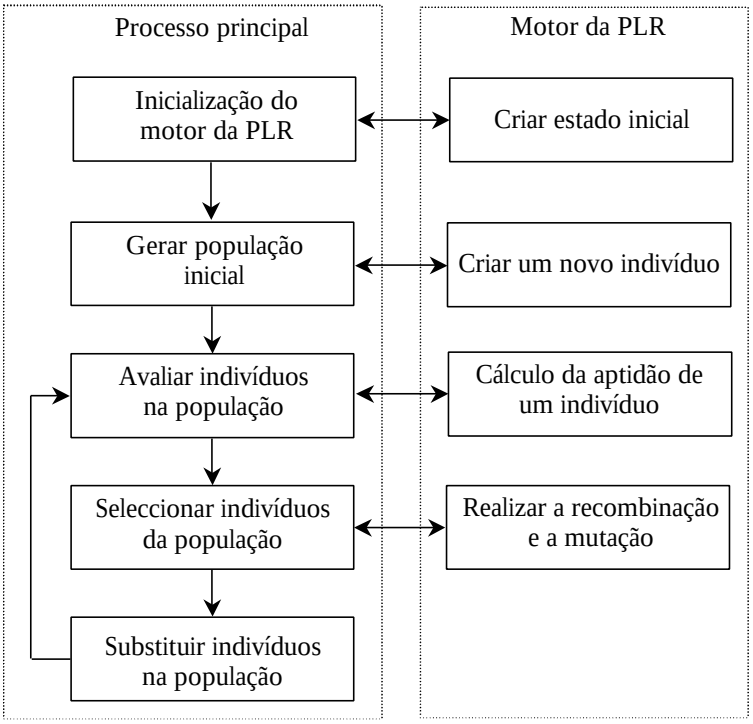


Figura 7-4: As duas partes principais do *LayGeRL*.

Esta divisão do *LayGeRL* em duas partes (processo principal e motor da PLR) permite a implementação de um processo principal genérico, permitindo que o mesmo possa ser utilizado para qualquer tipo de problema, sendo apenas necessário desenvolver, em função do tipo de problema em questão, o código responsável pela criação do estado inicial, por criar novos indivíduos, por avaliar a aptidão dos indivíduos e por executar os operadores genéticos de recombinação e mutação. Naturalmente que este código deve obedecer a determinadas requisitos de interface. Estes requisitos podem ser os que se encontram definidos na secção 6.4.2.

7.2.2 Representação das Soluções e População Inicial

Atendendo a que o motor da PLR executa todos os operadores genéticos, a representação de soluções escolhida relaciona-se directamente com a organização

interna das variáveis de decisão usado pelo *LayGeRL*. Esta é idêntica à organização interna utilizada pelo *LaRLo* já descrita na secção 5.1, e consiste numa lista ($\{\Phi\}$) com um tamanho igual ao número total de instalações a dispor na planta. Recorde-se que cada elemento da lista está associado a uma instalação e possui a estrutura seguinte:

$$\Phi\left(i(c,k), r\left(X_{ck}, Y_{ck}, C_{ck}, L_{ck}, F\right)\right)$$

Como se mostrou na secção 5.1, esta estrutura associada a cada instalação, representada pela extensão do predicado Φ, define completamente a sua posição na planta bem como a sua forma por intermédio das variáveis de decisão respectivas.

A lista $\{\Phi\}$ é um dos componentes do estado inicial. O estado inicial possui ainda uma outra componente que consiste na função de avaliação das soluções, sendo esta representada por um termo linear. A construção deste termo linear, que é usado na avaliação das soluções, foi já descrita na secção 5.3.1. Deste modo, a face visível do estado inicial do motor da PLR é representada pelo seguinte predicado Π, onde $\{\Phi\}$ é a lista de todas as instalações a dispor na planta e t_{Custo} é o termo linear que representa a função de avaliação.

$$\Pi\left(\{\Phi\}, t_{Custo}\right)$$

De acordo com a metodologia *GeRL*, o estado inicial do motor da PLR na resolução de um dado problema é essencial para a criação de novos indivíduos e para a execução dos operadores genéticos. De facto, a representação das soluções que se propõe é uma representação directa e baseia-se na lista $\{\Phi\}$ presente no primeiro argumento do estado inicial. Esta lista $\{\Phi\}$ serve de molde para os cromossomas de todos os indivíduos que estarão presentes na população ao longo das gerações. Portanto, cada gene tem uma estrutura Φ que representa a informação genética da respectiva instalação. Relativamente à função de avaliação t_{Custo}, que é o segundo argumento do estado inicial, esta pode ter também alguma utilidade para os operadores genéticos, nomeadamente se estes consistirem num algoritmo específico de optimização local, embora, neste trabalho, o seu uso se destine essencialmente ao cálculo do valor de aptidão dos indivíduos.

Com base no termo t_{Custo}, que representa a função de avaliação, a determinação do valor de aptidão dos indivíduos é uma tarefa simples. Depois de se terem instanciado todas a variáveis de decisão presentes em $\{\Phi\}$ basta uma simples linha de código na linguagem de programação Prolog para se determinar o custo da respectiva solução:

```
Custo is T_custo.
```

em que *Custo* é o valor do custo da solução e *T_Custo* é o termo linear que representa a função de avaliação. No entanto, a aplicação dos operadores genéticos pode gerar indivíduos inconsistentes com as restrições do problema. Se for admissível a sua existência na população, o cálculo dos respectivos valores de aptidão terá de ser efectuado de outra forma. Em princípio, estes indivíduos deverão apresentar um valor de aptidão nulo ou bastante baixo, para que não sobrevivam por muito tempo. Deste modo, para tratar todas as situação é usado o procedimento da Figura 7-5. Refira-se que a extensão do predicado *maxdominio*/2 devolve no segundo argumento o maior valor presente no domínio da variável do primeiro argumento. Note-se que uma variável instanciada possui, implicitamente, um domínio de tamanho unitário. Portanto, quando é avaliada uma solução inválida, é retornado o limite superior do valor de custo que qualquer solução poderá apresentar.

```
plr_avaliacao( Individuo( _, t_Custo ), Custo ) ←
    Var = t_Custo,
    maxdominio( Var, Custo ).
```

Figura 7-5: O cálculo do custo das soluções.

Como o problema em questão é um problema de minimização, em que as melhores soluções são aquelas que possuem o menor custo, é necessário usar um função de conversão do valor de avaliação (f_c) no valor de aptidão (f_a). Uma possibilidade é a função de conversão dada pela expressão (7-1), sendo c um valor constante parametrizável ou calculado em função de todos os valores de avaliação dos indivíduos presentes na população.

$$f_a = c \times \frac{1}{f_c} \tag{7-1}$$

Considerando o primeiro argumento do estado inicial do motor da PLR é possível aplicar o método de hibridização de Banier e Brisset (1998) descrito na secção 6.4.1 na resolução de PPLI. Este método é, como se referiu, independente dos problemas e satisfaz o principal requisito presente na escolha do método tratado neste capítulo para a resolução de PPLI. Este requisito passa pela escolha de um método capaz de encontrar boas soluções e que, ao mesmo tempo, não necessite de explorar todo o espaço de soluções. No entanto, optou-se por uma solução alternativa que se baseia na metodologia *GeRL*, e que incorpora nos operadores genéticos algum conhecimento específico do domínio do problema em questão. De facto, a incorporação deste tipo de conhecimento relaciona-se com um dos princípios relativos às linhas gerais de orientação para a hibridização defendidas por Davis (1991).

De acordo com o método desenvolvido, os indivíduos presentes na população do AG são soluções completas para o problema. Cada uma destas soluções resulta da instanciação de todas as variáveis de decisão com um valor do seu respectivo domínio que satisfaz as restrições do problema. A geração de novos indivíduos para a população inicial assemelha-se, portanto, a um procedimento de etiquetagem padrão que utiliza heurísticas específicas de ordenação de variáveis e de valores.

Um dos requisitos para que um AG proporcione boas soluções consiste em obter uma população inicial que possua um elevado grau de diversidade. Para conseguir isso, os indivíduos da população inicial devem representar soluções uniformemente distribuídas por todo o espaço de soluções. No caso dos PPLI isto pode ser conseguido pela disposição aleatória das instalações na planta. O mecanismo de propagação assegura que as soluções geradas são consistentes com as restrições do problema. Um possível algoritmo responsável por criar novos indivíduos é recursivo e encontra-se representado na Figura 7-6. Repare-se que este algoritmo não é mais do que um simples procedimento de etiquetagem padrão. Em termos funcionais e em cada iteração é escolhida uma instalação de forma aleatória à qual é dada uma forma, escolhida também de forma aleatória. A escolha da instalação é efectuada pela extensão do predicado *remove_rnd*/3 e a escolha da forma é realizada com a extensão do predicado *nodominio*/2, usando a heurística de ordem de selecção de valores *rnd*, que impõe uma ordem de selecção de valores aleatória. Note-se que,

dado que a área tem um valor fixo, uma vez definido o comprimento (largura) por propagação a largura (comprimento) é automaticamente definida. Por fim a instalação escolhida é colocada na planta através da instanciação das suas variáveis relacionadas com as coordenadas e de acordo com a heurística de ordem de selecção de valores λ.

```
novo_individuo( Π( {Φ}, _ ), λ ) ←
   novo_individuo({Φ}, λ ).

novo_individuo( [ ], _ ).
novo_individuo( {Φ}, λ ) ←
   remove_rnd( ( _, r( X, Y, C, L, _ ) ), {Φ}, T ),
   nodominio( C, rnd ),    nodominio( L, rnd ),
   nodominio( X, λ ),      nodominio( Y, λ ),
   novo_individuo( T, λ ).
```

Figura 7-6: Algoritmo responsável por gerar novos indivíduos.

7.2.3 Operador de Recombinação

Como foi referido na secção anterior, os operadores genéticos desenvolvidos incorporam conhecimento específico do domínio do problema em questão. Para o operador de recombinação este conhecimento relaciona-se com a posição que cada uma das instalações ocupa na planta. Em termos funcionais este operador de recombinação começa numa primeira fase por construir dois conjuntos mutuamente disjuntos S_1 e S_2. Estes conjuntos contêm a identificação de instalações. Todas as instalações a dispor na planta devem estar ou identificadas em S_1 ou S_2. Estes conjuntos são, portanto, mutuamente exclusivos e complementares. O tamanho de cada um dos dois conjuntos é escolhido de forma aleatória, embora, obviamente, o tamanho de um dependa do tamanho do outro. Deste modo, os tamanhos de S_1 e S_2 são determinados, respectivamente, pela expressão (7-2)[4] e (7-3). Note-se que o tamanho de S_1 é sempre igual ou inferior ao tamanho de S_2 e que n corresponde ao número total de instalações a dispor na planta. A escolha das instalações indicadas em cada um dos conjuntos também é efectuada de forma aleatória.

$$|S_1| = max\left(2; aleatorio\left(1; \left\lfloor n/2 \right\rfloor\right)\right) \quad (7\text{-}2)$$

$$|S_2| = n - |S_1| \quad (7\text{-}3)$$

Estes dois conjuntos constituem os parâmetros de controlo para a recombinação dos dois indivíduos progenitores seleccionados para reprodução. Com estes parâmetros de controlo a recombinação é efectuada em três etapas para cada um dos descendentes.

1. Considerando um dos progenitores, são colocadas na planta todas as instalações identificadas no conjunto S_2. Estas são colocadas na mesma posição que a definida pelo progenitor considerado. O formato destas instalações também é mantido. Isto corresponde a copiar os genes relativos às instalações identificadas em S_2 do progenitor considerado para o descendente. Dado que o progenitor representa uma solução consistente então a solução parcial resultante representada pelo descendente também é consistente;

2. Considera-se o conjunto S_1 e o outro progenitor e coloca-se apenas as instalações identificadas em S_1 na mesma posição que a definida pelo progenitor se o espaço que ocupam nessa posição estiver ainda disponível. A colocação de cada uma das instalações identificadas em S_1 também está dependente de ser possível garantir a consistência com as restrições do problema para que estas possam ser colocadas na planta;

3. Nesta última etapa, as instalações que ainda não foram colocadas na planta, quer por não existir espaço suficiente disponível no local pretendido ou a sua colocação originar inconsistência com uma ou mais restrições do problema, são colocadas nas regiões da planta que permanecem vazias e que permitam a construção de uma solução consistente.

Nestas duas últimas etapas, o formato das instalações é definido de acordo com o formato que possuem no segundo progenitor. A última etapa usa um

[4] A função *aleatório*(*n*, *m*) devolve um valor inteiro gerado aleatoriamente no intervalo de *n* a *m*.

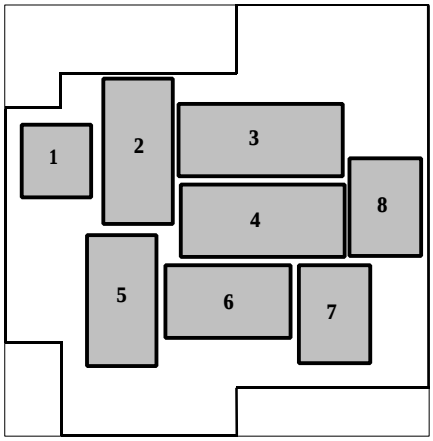
procedimento de etiquetagem típico da PLR(*DF*) de modo a que a solução seja consistente. A construção dos dois indivíduos resultantes da operação de recombinação é efectuada para cada um deles com o algoritmo da Figura 7-7. Recorde-se que a operação de recombinação, tal como propõe a metodologia *GeRL*, é efectuada geralmente com o procedimento indicado na Figura 6-16. Este utiliza a extensão do predicado *recombinacao*/4 para efectivamente realizar esta operação.

```
recombinacao( Π( {Φ}, _ ), [ S1, S2, λ ], P1, P2 ) ←
   recomb_etapa_1( {Φ}, S2, P1 ),
   recomb_etapa_2( {Φ}, S1, P2, [ ], Resto ),
   recomb_etapa_3( {Φ}, Resto, λ, P2 ).

recomb_etapa_1( _, [ ], _ ).
recomb_etapa_1( {Φ}, [ S|Ss ], P ) ←
   membro( ( S, R ), {Φ} ),    membro( ( S, R ), P ),
   !,
   recomb_etapa_1( {Φ}, Ss, P ).
recomb_etapa_1( {Φ}, [ _|Ss ], P ) ←
   recomb_etapa_1( {Φ}, Ss, P ).

recomb_etapa_2( _, [ ], _, Falha, Falha ).
recomb_etapa_2( {Φ}, [ S|Ss ], P, Acc, Falha ) ←
   membro( ( S, R ), {Φ} ),    membro( ( S, R ), P ),
   !,
   recomb_etapa_2( {Φ}, Ss, P, Acc, Falha ).
recomb_etapa_2( {Φ}, [ S|Ss ], P, Acc, Falha ) ←
   recomb_etapa_2( {Φ}, Ss, P, [S|Acc], Falha ).

recomb_etapa_3( _, [ ], _, _ ).
recomb_etapa_3( {Φ}, [ S|Ss ], λ, P ) ←
   membro( ( S, ( _, r( X, Y, C, L, _ ) ) ), {Φ} ),
   membro( ( S, ( _, r( _, _, C, L, _ ) ) ), P ),
   nodominio( X, λ ),    nodominio( Y, λ ),
   recomb_etapa_3( {Φ}, Ss, λ, P ).
recomb_etapa_3( {Φ}, [ _|Ss ], λ, P ) ←
   recomb_etapa_3( {Φ}, Ss, λ, P ).
```

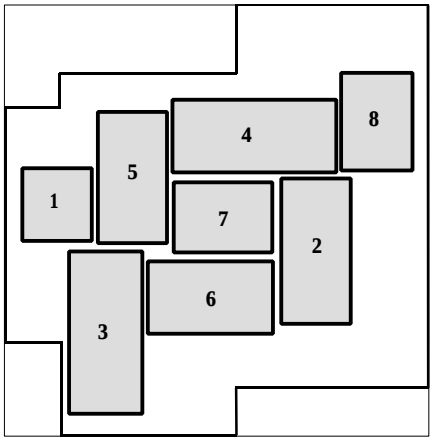
Figura 7-7: Algoritmo para o operador de recombinação.

De modo a, eventualmente, obter alguns ganhos de desempenho, é ainda possível considerar uma simplificação para este operador de recombinação. Esta simplificação passa por eliminar a segunda etapa do operador e considerar a colocação das instalações que eventualmente seriam colocadas nesta segunda etapa

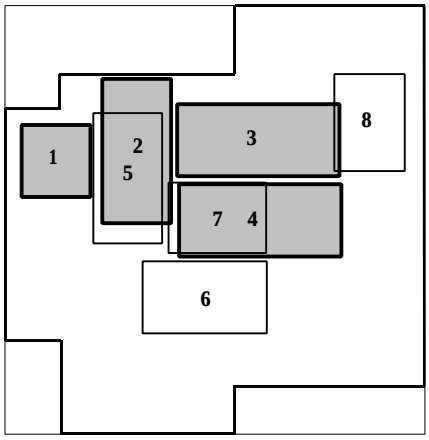
usando o procedimento de colocação das instalações da terceira etapa. A Figura 7-8 ilustra graficamente o operador de recombinação em acção considerando um hipotético PPLI com oito instalações sem nenhuma restrição particular.



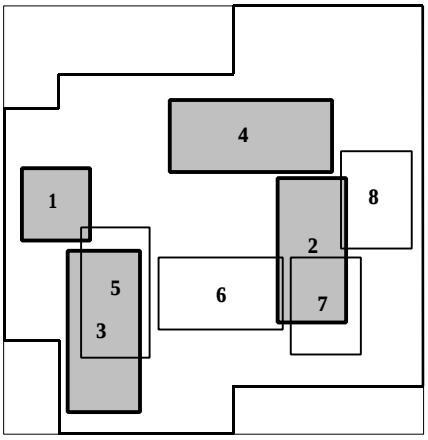
a) 1º progenitor



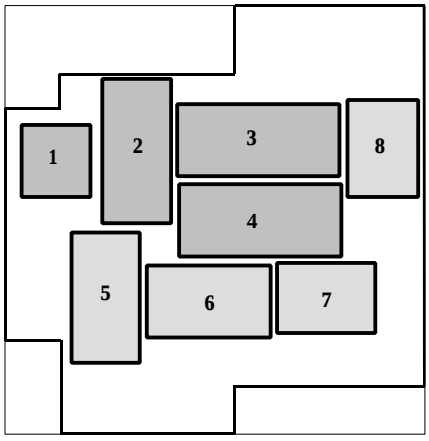
b) 2º progenitor



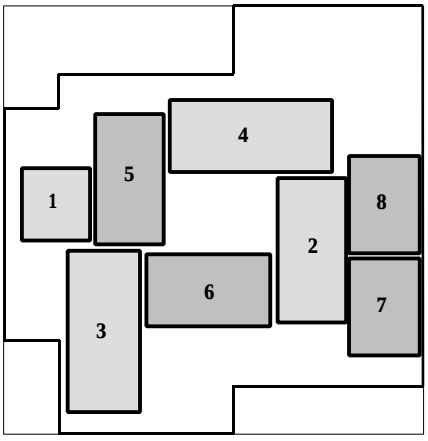
c) 1º descendente (solução inconsistente)



d) 2º descendente (solução inconsistente)



e) 1º descendente (solução reparada)



f) 2º descendente (solução reparada)

Figura 7-8: Um exemplo de um operador de recombinação.

Este operador de recombinação deve assegurar que as soluções resultantes representadas pelos descendentes devem ser consistentes com as restrições do problema. No entanto, em determinadas situações o operador de recombinação pode falhar na construção de um descendente. Estas situações ocorrem quando o operador não consegue dispor algumas das instalações identificadas em S_1 nas áreas disponíveis que restam na planta ou, então, pela colocação destas nas áreas disponíveis ser incompatível com as restrições do problema. Nesta situação de falha poderão ser consideradas três situações:

1. É atribuído um valor de aptidão muito baixo, ou mesmo nulo, a este 'indivíduo' descendente que representa uma solução inconsistente e, portanto, possui uma baixa probabilidade de sobreviver para as gerações seguintes;

2. O indivíduo descendente que representa uma solução inconsistente é substituído por um dos seus progenitores;

3. É criado um novo indivíduo que toma o lugar do indivíduo descendente que representa uma solução inconsistente.

Na primeira situação, se o indivíduo conseguir sobreviver para a geração seguinte, deve-se ter o cuidado de não o seleccionar para operações de cruzamento ou mesmo de mutação. A segunda situação é equivalente a introduzir na população uma cópia de um indivíduo já existente o que tem como principal inconveniente o de originar alguma perda de diversidade na população, e portanto, aumentando a tendência para o AG convergir para um mínimo local. Por último, na terceira situação, a introdução de um novo indivíduo na população tem como consequência o aumento da diversidade da população levando à exploração de diferentes regiões do espaço de soluções e, por este motivo, é esta a situação adoptada.

7.2.4 Operador de Mutação

Tal como o operador de recombinação, o operador de mutação também usa algum conhecimento específico do domínio do problema em questão. O operador de mutação altera o valor de um ou mais genes de um dado indivíduo. De certa forma, o

operador de recombinação descrito na secção anterior efectua uma forma específica de mutação. Esta forma específica de mutação consiste em alterar a posição de algumas instalações na planta. No entanto, é essencial, no processo de evolução, que de vez em quando a forma das instalações também se modifique. Esta alteração da forma das instalações é explicitamente realizada pelo operador de mutação. Entre diferentes possibilidades para este operador de mutação são consideradas as duas alternativas seguintes:

1. Escolher um valor diferente para o comprimento (largura) de uma instalação escolhida aleatoriamente. Esta alteração tem como efeito o de modificar a forma da instalação quando esta pode tomar diversas formas ou alterar sua a orientação na planta quando a sua forma é fixa;

2. Seleccionar aleatoriamente um conjunto de *n* instalações, sendo *n* um valor também aleatório e inferior ao número de instalações a colocar na planta. A forma de cada uma destas instalações escolhidas é modificada também de forma aleatória. Este operador também pode ser dividido em duas subcategorias, uma que requer que as instalações seleccionadas sejam adjacentes entre si e outra em que este requisito não seja obrigatório. Se as instalações são adjacentes entre si é ainda necessário decidir se mantêm as suas posições relativas na planta.

Para assegurar que as soluções sejam consistentes após a operação de mutação é normalmente necessário colocar estas instalações em locais diferentes das posições originais dado que as novas formas podem ser incompatíveis com área originalmente ocupada. Isto é também uma forma de mutação.

De acordo com as possibilidade referidas para o operador de mutação propõe-se um operador que, em termos funcionais, começa por determinar quantos e quais os genes que devem ser modificados. Estes genes, que serão modificados, são seleccionados aleatoriamente. Para isso é construído o conjunto *M* que identifica quais os genes (instalações) que serão modificados e que possui um tamanho dado pela expressão (7-4), onde *n* é o número de genes (instalações), e um conjunto *S* que é o conjunto complementar de *M* e cujo tamanho é dado pela expressão (7-5).

$$|M| = max\left(2; aleatorio\left(1; \left\lfloor n/2 \right\rfloor\right)\right) \tag{7-4}$$

$$|S| = n - |M| \tag{7-5}$$

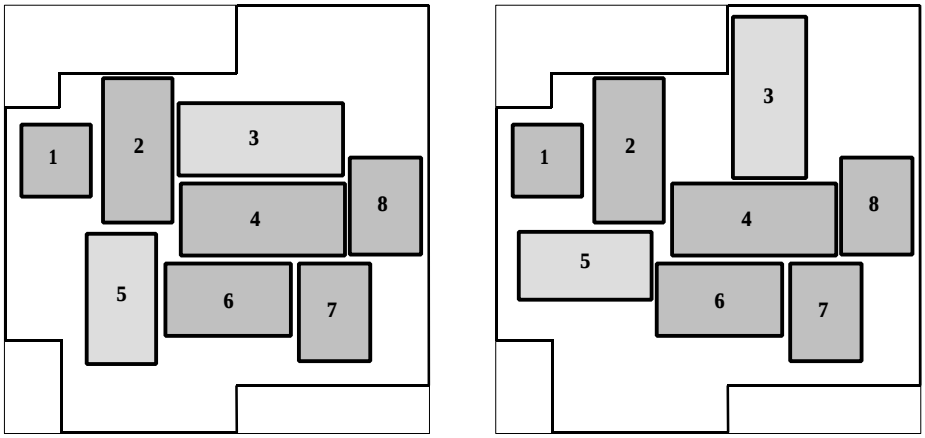
O conjunto *M* e o conjunto *S* constituem os parâmetros de controlo da operação de mutação. Com estes dois conjuntos, a operação começa por copiar todos os genes em *S* e de seguida termina com a colocação das instalações (genes) em *M* nas áreas da planta ainda disponíveis após a escolha aleatória de uma nova forma. A implementação do operador de mutação é feita de acordo com o algoritmo da Figura 7-9. Este algoritmo, representado pela extensão do predicado *mutacao*/3, é utilizado pelo procedimento da Figura 6-17 sempre que se realiza uma operação de mutação.

```
mutacao( Π( {Φ}, _ ), [ S, M, λ ], I ) ←
   mutacao_etapa_1( {Φ}, S, I ),
   mutacao_etapa_2( {Φ}, M, λ ).

mutacao_etapa_1( _, [ ], _ ).
mutacao_etapa_1( {Φ}, [ S | Ss ], I ) ←
   membro( ( S, R ), {Φ} ),
   membro( ( S, R ), I ),
   !,
   mutacao_etapa_1( {Φ}, Ss, I ).
mutacao_etapa_1( {Φ}, [ _ | Ss ], I ) ←
   mutacao_etapa_1( {Φ}, Ss, I ).

mutacao_etapa_3( _, [ ], _ ).
mutacao_etapa_3( {Φ}, [ S | Ss ], λ ) ←
   membro( ( S, ( _, r( X, Y, C, L, _ ) ) ), {Φ} ),
   nodominio( C, rnd ),
   nodominio( L, rnd ),
   nodominio( X, λ ),
   nodominio( Y, λ ),
   mutacao_etapa_3( {Φ}, Ss, λ ).
mutacao_etapa_3( {Φ}, [ _ | Ss ], λ ) ←
   mutacao_etapa_3( {Φ}, Ss, λ ).
```

Figura 7-9: Algoritmo para o operador de mutação.

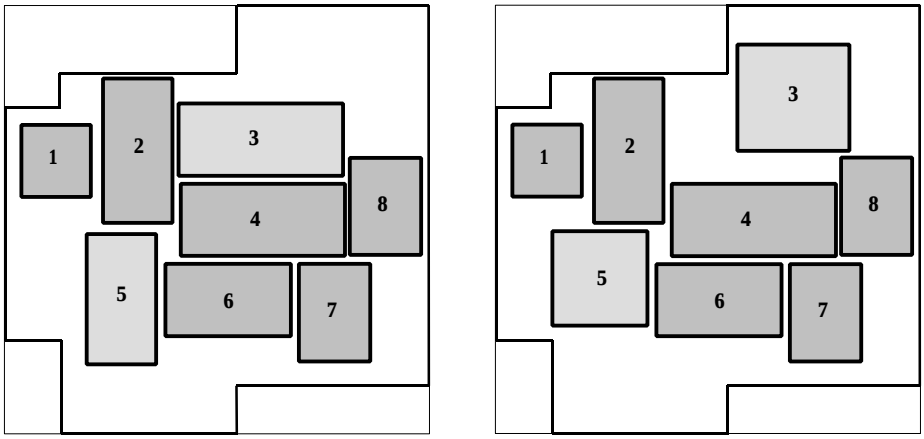
A Figura 7-10 mostra um exemplo do resultado de um operador de mutação que modifica dois genes. Este modificou a orientação na planta das instalações correspondentes, tendo também alterado as suas posições na planta. A Figura 7-11

mostra o mesmo exemplo, mas neste caso, é a forma das duas instalações que é modificada.



Indivíduo Indivíduo após a Mutação

Figura 7-10: Um exemplo de uma operação de mutação com alteração de orientação de algumas instalações.



Indivíduo Indivíduo após a Mutação

Figura 7-11: Um outro exemplo de uma operação de mutação com alteração do formato de algumas instalações.

Os indivíduos resultantes de operações de mutação devem também representar soluções consistentes com as restrições do problema, tal como acontece com o operador de recombinação. Portanto este operador de mutação também é susceptível de falhar ao realizar uma mutação, devido aos mesmos motivos que foram apontados para o operador de recombinação. Assim, em caso de falha, poderão ser consideradas as mesmas três situações identificadas atendendo às vantagens e inconvenientes que estas apresentam.

7.2.5 Os Operadores Genéticos e os Diferentes Tipos de Restrições

Os operadores genéticos de recombinação e mutação descritos foram concebidos tendo em conta, fundamentalmente, os problemas sem quaisquer requisitos especiais e onde as restrições de 'não sobreposição' e as restrições relacionadas com a forma das instalações são as únicas presentes. As experiências efectuadas com alguns problemas sem qualquer tipo de restrições específicas, de que se darão conta dos resultados em secções seguintes, mostraram um número de falhas relativamente pequeno (inferir a 5% relativamente ao número total operações de recombinação e de mutação) realizadas. Recorde-se que as falhas ocorrem quando os operadores genéticos não conseguem construir soluções consistentes com as restrições do problema em questão. Verificou-se também existir uma relação entre este número de falhas com a razão entre a área da planta e área total necessária para dispor todas as instalações. É óbvio que, para as mesmas instalações, quanto maior for a área da planta, maior é a possibilidade de um operador genético ter sucesso e menor será o esforço computacional para encontrar uma solução consistente com as restrições do problema.

Ao serem adicionadas outros tipos de restrições para expressar requisitos específicos relativamente à instancia do problema a solucionar, e de acordo com algumas experiências efectuadas, verificou-se um aumento do número de falhas. É claro que as restrições adicionais limitam ainda mais o espaço de soluções, obrigando a um maior esforço computacional para encontrar soluções viáveis. Atendendo à forma como os operadores genéticos operam, verifica-se que este aumento do número de falhas está relacionado com determinadas restrições binárias, nomeadamente as restrições de distância, de vizinhança e adjacência. Para limitar o efeito que estes tipos de restrições têm sobre o número de falhas, é imposta uma condição relativamente à forma como os operadores genéticos seleccionam as instalações. Esta condição obriga a que, quando uma instalação é seleccionada para um dos conjuntos que definem os parâmetros de controlo do operador, as outras instalações que se relacionam com a primeira, através de uma destas restrições binárias dos tipos que foram referidos, devem estar também presentes nesse

conjunto. Verificou-se que, ao se considerar esta condição, o aumento do número de falhas é inferior.

Por último, verifica-se que, se a ordem com que as instalações são colocadas na planta tiver também em consideração os tipos de restrições binárias referidos, o desempenho na construção das novas soluções, não só por intermédio do operadores genéticos, mas também na criação de novos indivíduos, apresenta algumas melhorias. Este facto leva a que duas instalações que participam numa restrição binária sejam consecutivas na ordem em que são colocadas na planta. Em situações de conflito é dada prioridade às restrições de adjacência, depois às restrições de vizinhança, e por último às restrições de distância.

7.2.6 Problemas de Planeamento de Espaço

Um dos requisitos fundamentais que estiveram presentes na concepção dos operadores genéticos propostos passou por dota-los de um desempenho tão bom quanto possível. Este requisito não foi fácil de satisfazer, principalmente se se atender aos resultados obtidos com o *LaRLo*, indicados na secção 5.3, relativos ao tempo de processamento que foi gasto para encontrar a primeira solução. Considerando um AG típico que possui uma população de 100 indivíduos e que a evolução decorre ao longo de 100 gerações, esta poderá em cada geração realizar no máximo 50 recombinações gerando 100 descendentes. Se em média cada operação de recombinação necessitar de um tempo de processamento igual a 1 segundo, sem atender ao operador de mutação, o AG só terminaria ao fim de aproximadamente uma hora. Considerando que o *LaRLo* na resolução do problema 'ppli15' demorou em média mais de 10 segundos para encontrar a primeira solução é de esperar que um AG com as características descritas necessite de aproximadamente 20 horas.

A primeira versão do protótipo do *LayGeRL*, embora mostrando um desempenho melhor do que as expectativas iniciais, atendendo ao desempenho do *LaRLo*, apresentou um desempenho relativamente fraco relativamente ao desejado. Há que salientar que a solução encontrada para um dado PPLI em concreto, depois de implementada, deverá estar operacional, em princípio, ao longo de alguns anos e, deste modo, é razoável dispensar algumas horas, ou mesmo dias, na sua resolução.

De qualquer forma, para melhorar o desempenho do *LayGeRL*, verificou-se que grande parte do tempo de processamento é gasto no mecanismo de propagação e em retrocessos devido a tentativas de atribuição de valores às variáveis de decisão do seu respectivo domínio e que originam ramos na árvore de pesquisa que não levam a soluções consistentes com as restrições do problema. Para minimizar este tempo gasto com o mecanismo de propagação e com os retrocessos, é necessário em primeiro lugar constatar que o PPLI é, essencialmente, um problema de planeamento de espaço a duas dimensões. No formalismo da PLR(*DF*) o domínio das variáveis é unidimensional e geralmente discreto. O uso de variáveis que possuem domínios unidimensionais pode não ser a forma mais adequada para solucionar os problemas de planeamento espacial.

Assim, com o objectivo de melhorar o desempenho, foi desenvolvido um método que usa informação espacial bidimensional permitindo, assim, tratar alguns dos problemas de planeamento de espaço de um modo mais natural. De uma forma simplificada, este método consiste em colocar um conjunto de objectos rectangulares $\{R\}$ dentro de um outro objecto rectangular P maior, denominado por planta. Os objectos são colocados um a um e, em cada instante, existe um registo do espaço disponível em P para os objectos por colocar. Este registo do espaço disponível em P pode ser considerado como uma espécie de domínio espacial que tem a seguinte forma:

$$[\, r_1, ..., r_i, ..., r_k \,].$$

Este domínio espacial consiste numa lista de rectângulos que se podem sobrepor e que indicam regiões em P livres para colocar os objectos $\{R\}$. Sempre que um objecto é colocado em P este domínio espacial é actualizado. Este método considera o domínio espacial associado a P e não a cada um dos objectos $\{R\}$. A Figura 7-12 mostra um caso em que domínio espacial de P, após a colocação do objecto R_1, é $[r_1, r_2, r_3, r_4]$. A partir deste momento a colocação de R_2 em P só é possível se ocorrer dentro de um dos rectângulos que definem o domínio espacial de P. O algoritmo responsável pela actualização do domínio espacial é o algoritmo *rever* indicado na Figura 7-13. Em função do domínio espacial de P (*Dom*) e da região que o objecto R irá ocupar, representada por $r(x,y,c,l)$, este devolve o domínio

espacial que resulta da colocação de *R*. Os rectângulos do domínio espacial cujos comprimento ou largura sejam inferiores a, respectivamente, *Cmin* e *Lmin* não são considerados no domínio espacial resultante. Este algoritmo retorna um domínio espacial que possui o número mínimo de rectângulos que definem completamente o domínio, o que significa que nenhum rectângulo presente no domínio está incluído noutro.

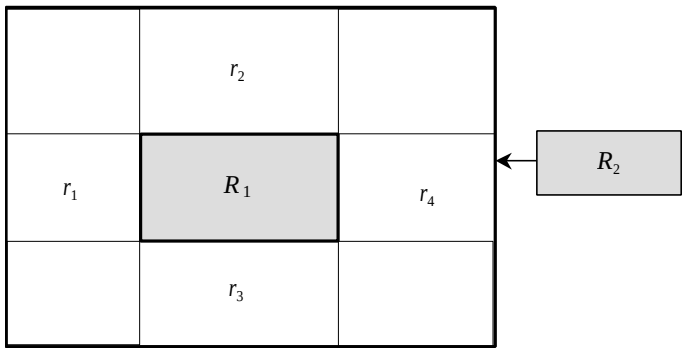


Figura 7-12: O domínio espacial de uma planta após a colocação de R_1 e que define quais as regiões admissíveis para a colocação de R_2.

procedimento *rever*(*Dom*, *r*(*x*, *y*, *c*, *l*), *Cmin*, *Lmin*, *NovoDom*)
  LISTA lst ← ∅
  para cada *r*(*X*, *Y*, *C*, *L*) ∈ *Dom*
    acima ← *r*(*X*, *max*(*y*+*l*, *Y*), *W*, *min*(*Y*+*L*-*y*-*l*, *L*))
    abaixo ← *r*(*X*, *Y*, *W*, *min*(*y*-*Y*, *L*))
    esquerda ← *r*(*X*, *Y*, *min*(*x*-*X*, *W*), *L*)
    direita ← *r*(*max*(*x*+*w*,*X*), *Y*, *min*(*X*+*W*-*x*-*w*, *W*), *L*)
    se *acima.c* ≥ *Cmin* ∧ *acima.l* ≥ *Lmin* **então** *lst* ← {*lst* ∩ *acima*}
    se *abaixo.c* ≥ *Cmin* ∧ *abaixo.l* ≥ *Lmin* **então** *lst* ← {*lst* ∩ *abaixo*}
    se *esquerda.c* ≥ *Cmin* ∧ *esquerda.l* ≥ *Lmin* **então** *lst* ← {*lst* ∩ *esquerda*}
    se *direita.c* ≥ *Cmin* ∧ *direita.l* ≥ *Lmin* **então** *lst* ← {*lst* ∩ *direita*}
  fim
  NovoDom ← ∀ *r* ∈ *lst* , ~∃ *r'* ∈ *lst* : *r* ⊂ *r'*
fim *rever*

Figura 7-13: Algoritmo *REVER* para realizar a actualização do domínio espacial.

Note-se que, em última análise, esta método poderá evoluir para um meta-interpretador de restrições particular em que cada variável é representada por

um objecto rectangular e em que o domínio de cada uma é um domínio espacial. As restrições estabelecem relações entre objectos que devem ser satisfeitas.

Como se deve ter verificado, sempre que um objecto rectangular *R* é colocado em *P*, a actualização do domínio espacial com o algoritmo *rever* constitui um forma implícita da restrição de 'não sobreposição'. Embora se possam considerar outros tipos de restrições, é apenas esta forma da restrição ('não sobreposição') que se considera no método. De acordo com o modelo adoptado para os PPLI, é este tipo de restrições que possui o maior peso em relação ao número total de restrições a tratar pelo mecanismo de propagação de restrições.

A utilização do método no *LayGeRL* considera que cada instalação corresponde a um objecto rectangular e a planta fabril é a planta *P* onde se dispõem as instalações. Quando a planta fabril, representada por *P*, não possui uma forma rectangular é necessário construir o domínio espacial inicial da planta pela exclusão das áreas que não pertencem à planta do rectângulo que a envolve. Como já foi referido, considera-se que estas áreas possuem uma forma rectangular. Deste modo, o domínio espacial inicial é determinado, com o auxilio do algoritmo *rever*, pela "colocação" de objectos rectangulares, que representam estas áreas, no rectângulo que envolve a planta fabril real. A Figura 7-14 mostra um exemplo de uma planta com uma forma não rectangular onde se pretendem colocar as instalações I_1 e I_2.

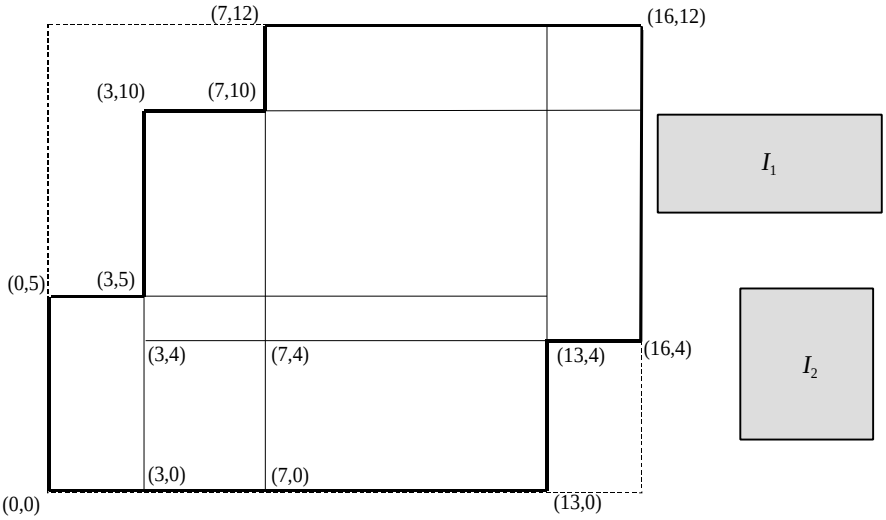


Figura 7-14: O domínio espacial inicial duma planta *P* onde se pretende colocar I_1 e I_2.

De acordo com o exemplo da Figura 7-14, o uso sucessivo do algoritmo *rever* para cada uma das regiões não pertencentes à planta resulta no domínio espacial inicial de *P* seguinte

[*r*(0, 0, 13, 5), *r*(3, 4, 13, 6), *r*(7, 4, 9, 8), *r*(3, 0, 10, 10), *r*(7, 0, 6, 12)].

Colocando de seguida a instalação I_1 em *P*, tal como se ilustra na Figura 7-15, resulta que o domínio espacial de *P* actualizado passa a ser o seguinte, considerando que são admitidas instalações que possuem valores para o comprimento e/ou para a largura iguais ou superiores a 1 unidade.

[*r*(0, 0, 13, 3), *r*(0, 0, 5, 5), *r*(12, 0, 1, 5), *r*(3, 6, 13, 4), *r*(3, 4, 2, 6), *r*(12, 4, 1, 6), *r*(7, 6, 9, 6), *r*(12, 4, 4, 8), *r*(3, 0, 2, 10), *r*(3, 0, 10, 3), *r*(3, 6, 10, 4), *r*(12, 0, 1, 10), *r*(7, 0, 6, 3), *r*(7, 6, 6, 4), *r*(12, 0, 1, 12)].

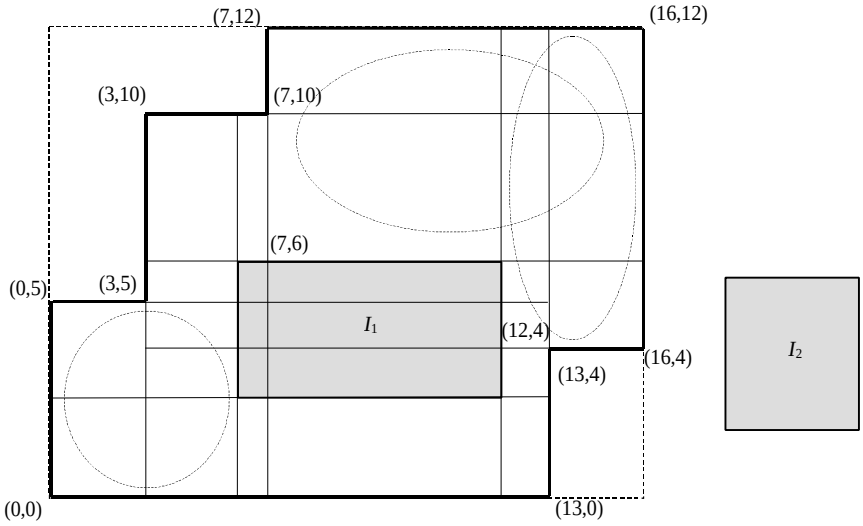


Figura 7-15: O domínio espacial duma planta *P* onde se pretende colocar o objecto I_2 após a colocação de I_1.

Admitindo que I_2 pode tomar unicamente a forma que é indicada na Figura 7-15, então a sua colocação em *P* considera apenas as áreas *r*(0, 0, 5, 5), *r*(7, 6, 9, 6), *r*(12, 4, 4, 8) do domínio espacial de *P* indicadas na figura por uma oval, uma vez que as restantes não podem acomodar I_2

A construção dos indivíduos, tanto na criação de novos indivíduos como na execução dos operadores genéticos, requer que a instanciação das variáveis de decisão relativas a cada instalação, que seria efectuado usando simplesmente a

extensão do predicado *nodominio*/2 para cada uma das variáveis, passe a ser efectuada de acordo com o algoritmo indicado na Figura 7-16 que é representado pela extensão do predicado *rect_nodominio*/4. Repare-se que este começa por escolher a forma para a instalação e de seguida exclui as áreas que não podem acomodar esta instalação com a forma escolhida por via da extensão do predicado *rectangulos_validos*/4. O passo seguinte passa por limitar a posição da instalação a uma das restantes regiões. Este processo termina quando a instalação é colocada na planta.

```
rect_nodominio( r( X, Y, C, L, _ ), λ, Rects, Novos_Rects ) ←
   forma( C, L ),
   rectangulos_validos( Rects, 2 * C, 2 * L, Validos ),

   membro( r( Xd, Yd, Cd, Ld ), Validos ),
   X ≥ Xd + C,    X ≤ Xd + Cd - C,
   Y ≥ Yd + L,    Y ≤ Yd + Ld - L,
   dispor( [X, Y], λ ),

   rever( Rects, r( X - C, Y - L, 2 * C, 2 * L ), 1, 1, Novos_Rects ).


forma( C, L ) ←
   nodominio( C, rnd ),
   nodominio( L, rnd ),
   !.


dispor( X, Y, λ ) ←
   nodominio( X, λ ),
   nodominio( Y, λ ),
   !.
```

Figura 7-16: Algoritmo para dispor uma instalação na planta fabril usando o domínio espacial.

Dadas as características do domínio espacial existe sempre a possibilidade de a mesma posição ser explorada duas ou mais vezes. Isto acontece porque podem surgir áreas que são comuns a diferentes rectângulos do domínio espacial e que permitem acomodar as instalações. Dado que é satisfatório obter a primeira solução completa na execução dos operadores genéticos, esta situação acaba por, em termos práticos, não se revelar de grande importância. Já na aplicação deste método ao *LaRLo*, esta situação reveste-se de grande importância, dado que pode ter efeitos nefastos no desempenho, atendendo à forma como o espaço de soluções é explorado.

7.3 *LayGeRL* na Resolução dos Casos de Teste

Esta secção destina-se a discutir alguns dos aspectos relacionados com a forma como o *LayGeRL* explora o espaço de soluções e a avaliar o seu desempenho em termos da qualidade das soluções proporcionadas em comparação com o *LaRLo*.

7.3.1 Exploração do Espaço de Soluções

O *LayGeRL* é um protótipo de uma aplicação baseada na metodologia *GeRL* para solucionar PPLI. A implementação do módulo relativo ao motor da PLR foi escrita para o meta-interpretador de PLR(*DF*) *ECLiPSe* 4.2 (Schimpf *et al*, 1999) e o módulo do processo principal é baseado no *GALib* (Wall, 1996).

O modelo de evolução escolhido para o AG usado no *LayGeRL* é um modelo de evolução intermédio que se situa entre o AG simples (Holland, 1975) e o AG de estado estacionário (Whitley, 1989). Em cada geração é seleccionada uma determinada percentagem de indivíduos da população para reprodução. Os descendentes resultantes são então inseridos na população por troca com os seus piores indivíduos. Esta percentagem de indivíduos substituídos na população em cada geração, denominada de taxa de substituição, é um dos parâmetros dos AG que usam este modelo de evolução. A selecção dos indivíduos, para a escolha dos progenitores para reprodução, é efectuada de acordo com a técnica de amostragem estocástica da roleta. A conversão dos valores de aptidão em valores de selecção é realizada de acordo com o método de conversão de escala linear.

Para parar a exploração do espaço de soluções, definiu-se um critério de paragem baseado em dois parâmetros. Um dos parâmetros é o número máximo de gerações em que a evolução deve ocorrer e o outro baseia-se no desvio padrão dos valores de aptidão dos indivíduos na população. A evolução termina quando, em primeiro lugar, a contagem das gerações for igual ou superior ao número máximo de gerações (considerou-se um número máximo de gerações igual a 20 vezes o número de genes) ou quando o desvio padrão dos valores de aptidão for inferior a 0,01% do custo do melhor indivíduo na população.

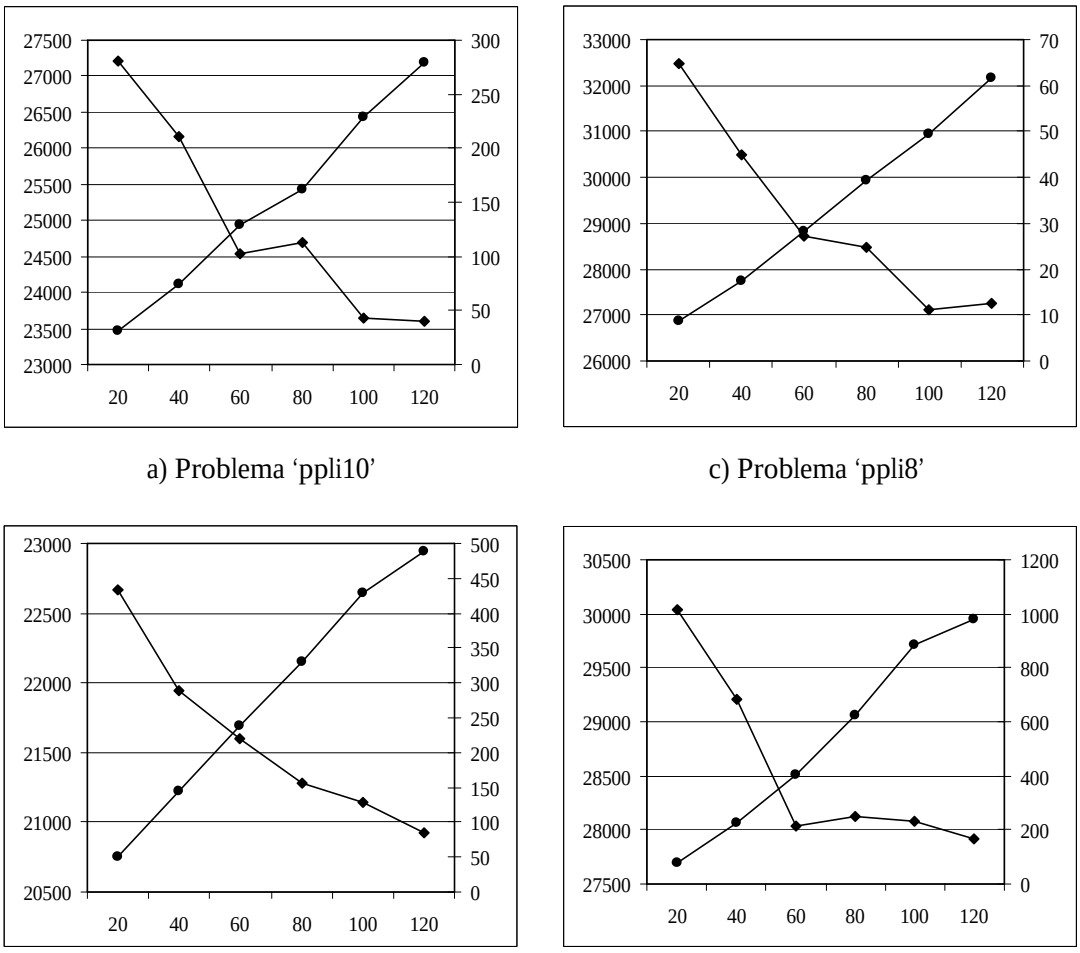
Os operadores genéticos implementados, de recombinação e mutação, seguem,

respectivamente, as especificações indicadas nas secções 7.2.3 e 7.2.4. Estes usam informação espacial de acordo com o método descrito na secção 7.2.6 para obter um melhor desempenho. O uso de informação espacial obriga a que as variáveis de decisão associadas a cada gene (ou instalação) sejam instanciadas de acordo com o algoritmo indicado na Figura 7-16. Um dos parâmetros deste algoritmo consiste na heurística para a ordem de selecção de valores λ. Depois de se efectuarem algumas experiências com diversos casos de teste verificou-se que a heurística que proporciona um melhor desempenho é a que selecciona em primeiro lugar os valores nas extremidades do domínio e depois alterna até ao valor médio. Considerando, por exemplo, o domínio 1..10, a ordem de selecção de valores de acordo com esta heurística é {1, 10, 2, 9, 3, 8, 4, 7, 5, 6}.

7.3.2 Influência dos Principais Parâmetros na Qualidade das Soluções

Para determinar qual a melhor gama de valores para cada um dos principais parâmetros do AG que potencialmente permitem obter melhores resultados, tendo em conta o modelo de evolução escolhido e os operadores genéticos desenvolvidos, foram realizadas algumas experiências. Estes parâmetros são o tamanho da população, a taxa de substituição, a probabilidade de recombinação e a probabilidade de mutação. Estas experiências podem ser agrupadas em quatro categorias, de acordo com a influência que cada um dos parâmetros tem na qualidade das soluções. Deve referir-se que a melhor gama de valores obtida com as experiências para cada um dos parâmetros deve ser considerada meramente indicativa dado que a determinação do melhor valor para cada um dos parâmetros não é uma tarefa trivial. Os casos de teste escolhidos para estas experiências foram os problemas ‘ppli8’, ‘ppli10’ e ‘ppli15’. As principais características destes casos foram já indicadas na Tabela 5-3. Relativamente ao problema ‘ppli10’ foi também considerada a situação em que se colocam restrições de adjacência entre pares de instalações, escolhidos também de acordo com o método de determinação da correspondência de peso máximo descrito na secção 5.3.5. Cada um dos casos de teste considerados foram solucionados 10 (dez) vezes para cada combinação de valores destes parâmetros.

O primeiro parâmetro estudado foi a influência do tamanho da população na qualidade das soluções e no tempo de processamento gasto para os diferentes problemas. Os respectivos gráficos encontra-se indicados na Figura 7-17. Note-se que os gráficos apresentados nesta secção consideram os parâmetros analisados representados no eixo dos *xx*, o custo das soluções no eixo dos *yy* à esquerda e o tempo (em segundos) no eixo dos *yy* à direita.



a) Problema ‘ppli10’ c) Problema ‘ppli8’

b) Problema ‘ppli10’ (restrições de adjacência) d) Problema ‘ppli15’

Taxa de substituição 0,4, prob. recombinação 0,8 e prob. mutação 0,1.

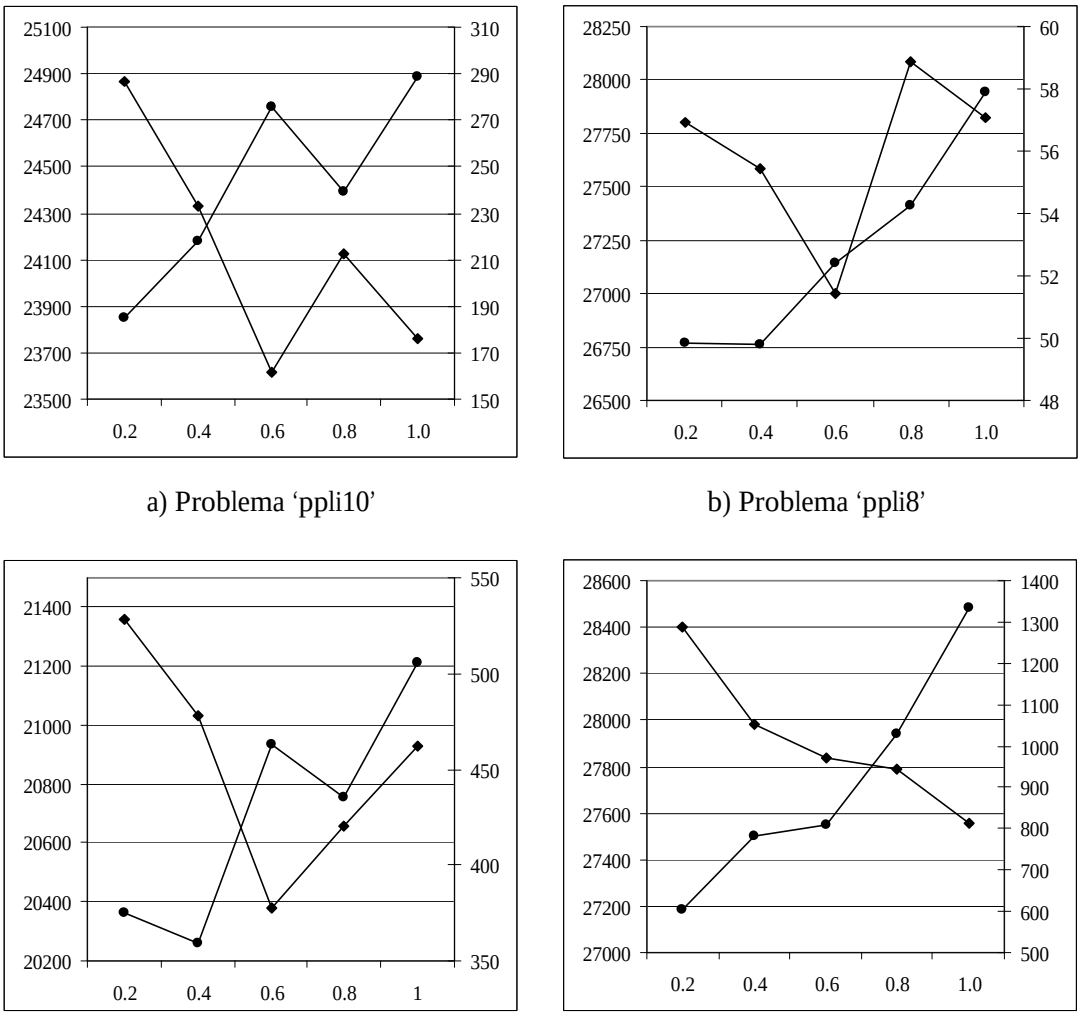
Custo Médio Tempo Médio

Figura 7-17: Evolução da qualidade das soluções e do tempo de processamento em função do tamanho da população.

Como seria de esperar o aumento do tamanho da população melhora a qualidade das soluções já que o seu custo é menor, mas, no entanto, esta melhoria é feita à custo de um aumento do tempo de processamento tal como seria de esperar.

Pela análise dos gráficos, é razoável considerar um bom compromisso o tamanho da população em torno dos 100 indivíduos.

O parâmetro seguinte é o valor da taxa de substituição. A relação que este parâmetro tem com a qualidade das soluções e com o tempo de processamento gasto para os diferentes problemas encontra-se representada nos gráficos Figura 7-18.



a) Problema ‘ppli10’ b) Problema ‘ppli8’

c) Problema ‘ppli10’ (restrições de adjacência) d) Problema ‘ppli15’

Tamanho da população 100, prob. recombinação 0,8 e prob. mutação 0,1.

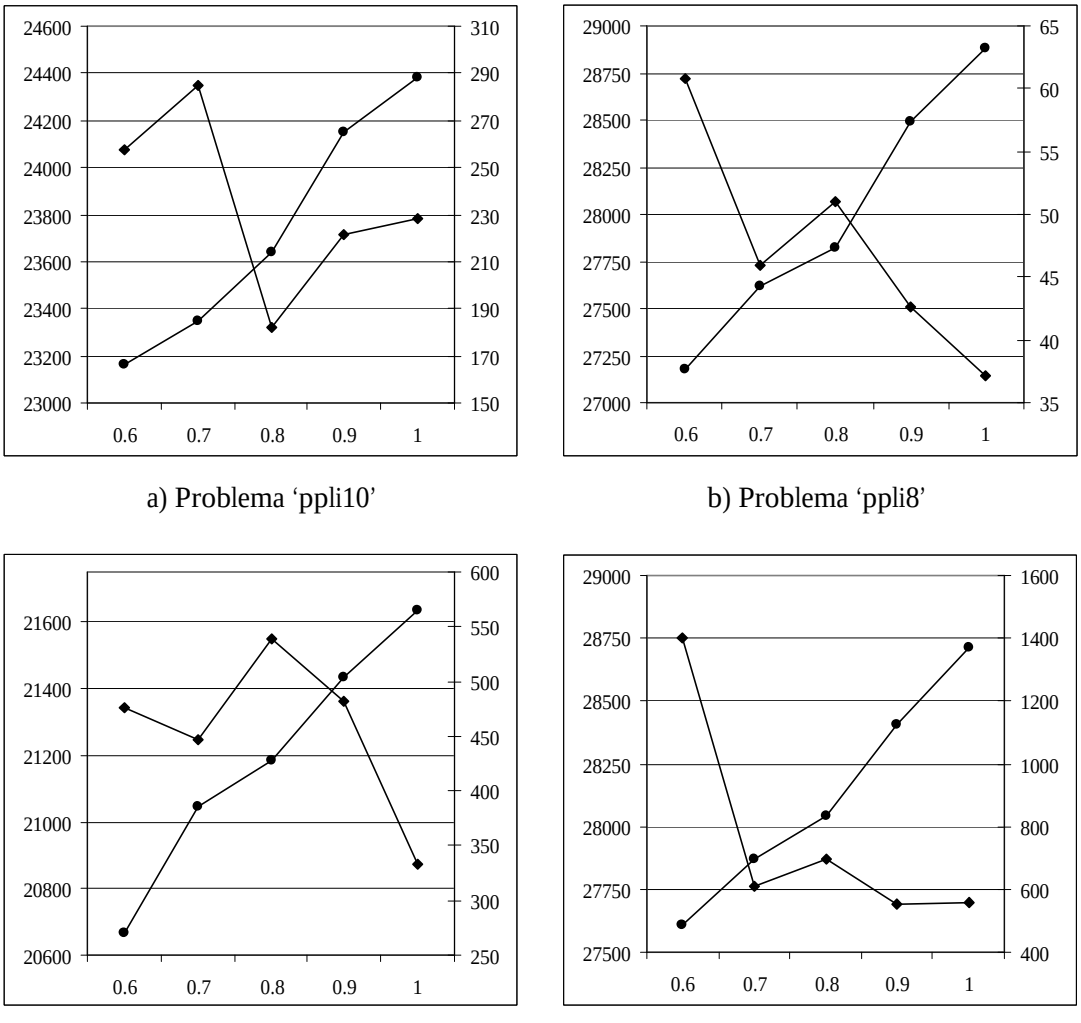
Custo Médio Tempo Médio

Figura 7-18: Evolução da qualidade das soluções e do tempo de processamento em função da taxa de substituição.

Observa-se neste caso que o tempo de processamento aumenta com o aumento da taxa de substituição, o que é lógico dado que se realizam mais operações genéticas

enquanto que o custo das soluções decresce. No entanto, como se pode verificar, a partir de um valor da taxa de substituição superior 0,6 se possa registar uma inversão na tendência de diminuição do custo das soluções. Este facto pode estar relacionado com o número de indivíduos substituídos por descendentes de pior qualidade.

A probabilidade de recombinação é outro parâmetro com influência na qualidade das soluções. Esta influência, para os problemas solucionados, encontra-se ilustrada nos gráficos da Figura 7-19.



a) Problema ‘ppli10’

b) Problema ‘ppli8’

c) Problema ‘ppli10’ (restrições de adjacência)

d) Problema ‘ppli15’

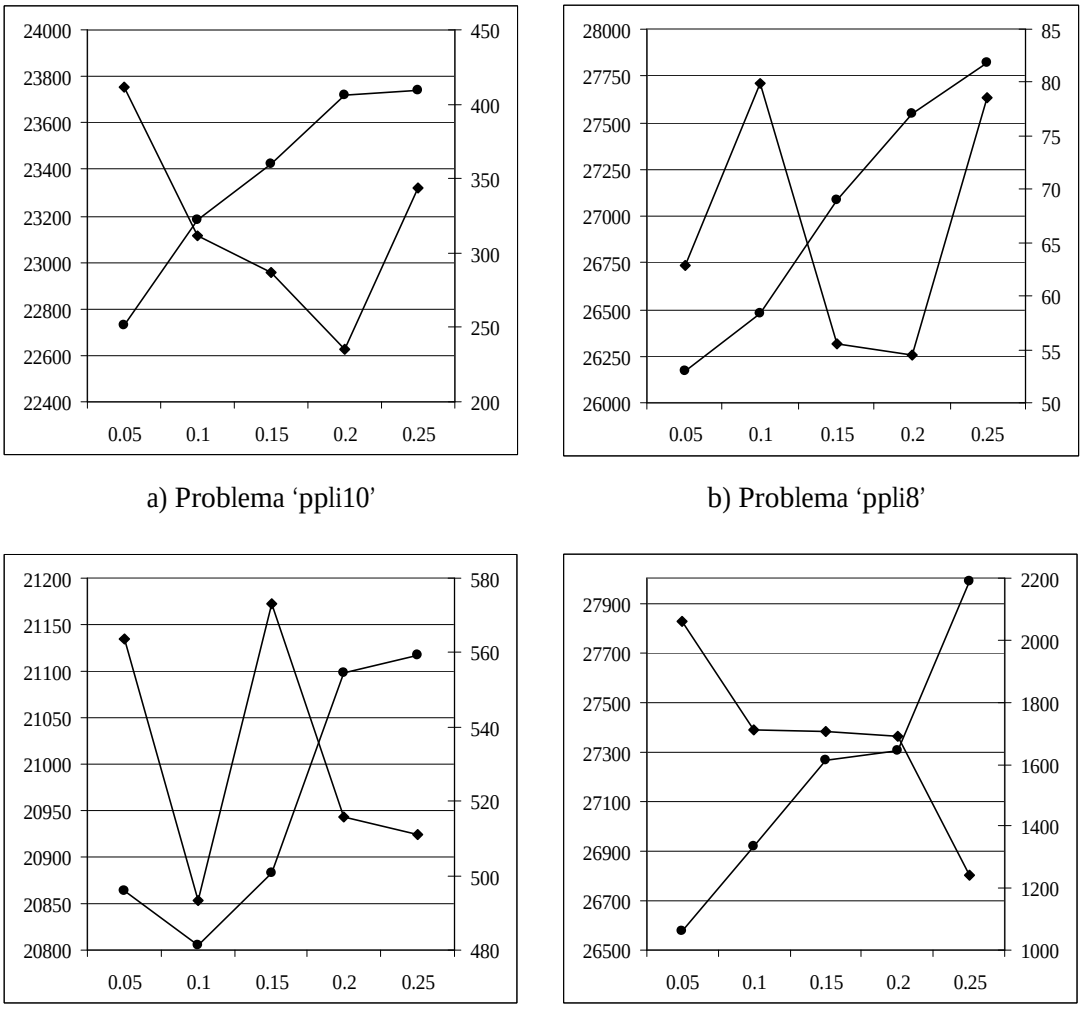
Tamanho da população 100, taxa de substituição 0,4 e prob. mutação 0,1.

Custo Médio Tempo Médio

Figura 7-19: Evolução da qualidade das soluções e do tempo de processamento em função da probabilidade de recombinação.

Como já seria de esperar, verifica-se que tempo de processamento aumenta com o aumento da probabilidade de recombinação tal como em geral a qualidade das soluções. Deve-se referir que o caso de teste 'ppli10', em que não são incluídas restrições de adjacência, foi uma excepção. Neste caso, a qualidade das soluções mostrou uma ligeira tendência para piorar para valores da probabilidade de recombinação superiores a 0,8.

Por último, surge a probabilidade de mutação. A sua influência na qualidade das soluções encontra-se representada na Figura 7-20.



a) Problema 'ppli10'

b) Problema 'ppli8'

c) Problema 'ppli10' (restrições de adjacência)

d) Problema 'ppli15'

Tamanho da população 100, prob. recombinação 1,0 e prob. mutação 0,1.

Custo Médio Tempo Médio

Figura 7-20: Evolução da qualidade das soluções e do tempo de processamento em função da probabilidade de mutação.

Verifica-se também que o aumento da probabilidade de mutação faz aumentar o tempo de processamento, o que já era previsível. Há que referir, no entanto, que para a maior parte dos problemas este aumento do tempo de processamento não é significativo, o que não acontece com este caso. Este facto relaciona-se com as características do operador de mutação usado, que não é mais do que uma forma específica de um procedimento de etiquetagem em PLR(*DF*).

Já a relação existente entre a probabilidade mutação e a qualidade das soluções observa-se que não é tão clara. No entanto, parece existir uma tendência para as melhores soluções surgirem no intervalo de 0,1 a 0,2.

A Tabela 7-1 resume qual o intervalo de valores para os quatro parâmetros analisados que mostraram, pelas experiências realizadas, uma tendência para proporcionarem soluções de melhor qualidade.

Tabela 7-1: A gama de valores para os parâmetros do AG do *LayGeRL*.

Tamanho da população	80 .. 100
Taxa de substituição	0,4 .. 0,8
Probabilidade de recombinação	0,8 .. 1,0
Probabilidade de mutação	0,1 .. 0,2

Estes resultados não permitem concluir definitivamente qual a melhor gama de valores para os quatro parâmetros aqui analisados, mas permitem a observação de uma tendência para a gama de valores que permite obter soluções de melhor qualidade. De qualquer forma, a escolha dos melhores valores para os diferentes parâmetros depende muitas das vezes do problema a solucionar, da sua dimensão e complexidade e também do tempo de processamento que se pode dispensar. É muitas vezes necessário encontrar um bom compromisso entre os diferentes factores envolvidos.

7.3.3 Resultados para os Casos de Teste

De acordo com as experiências realizadas, para determinar a influência dos principais parâmetros do AG implementado no *LayGeRL* na qualidade das soluções,

foram solucionados os restantes casos de teste indicados na Tabela 5-3, tendo sido escolhidos valores para os diferentes parâmetros que, tendo em conta a dimensão e complexidade do problema, à partida proporcionam um bom compromisso entre a qualidade das soluções e o tempo de processamento.

Os parâmetros que proporcionaram as melhores soluções para todos os casos de teste envolvidos nas experiência realizadas bem como os parâmetros escolhidos para os restantes casos de teste encontram-se indicados na Tabela 7-2.

Tabela 7-2: Os parâmetros usados para solucionar os diferentes casos de teste.

	ppli8		ppli10		ppli10c		ppli15		ppli24	
Adjacência	N	S	N	S	N	S	N	S	N	S
Tamanho da população	100	100	100	100	100	100	100	100	80	60
Taxa de substituição	0,4	0,4	0,4	0,5	0,5	0,5	0,4	0,5	0,1	0,1
Prob. de recombinação	1,0	1,0	1,0	0,9	0,9	0,9	1,0	0,9	0,8	0,8
Prob. de mutação	0.2	0,15	0,2	0,15	0,15	0,15	0,25	0,15	0,05	0,05

Tal como se pode verificar, cada caso de teste foi solucionado considerando o problema com e sem restrições de adjacência adicionais (colunas S e N respectivamente). A colocação de restrições de adjacência considera a escolha de pares de instalações de acordo com o método de determinação da correspondência de peso máximo que foi descrito na secção 5.3.5. Por outro lado, a resolução de cada caso de teste foi realizada 10 (dez) vezes, dado o carácter estocástico dos AG.

Os custos das soluções encontradas para os diferentes casos de teste, não considerando as restrições de adjacência, encontram-se indicados na Tabela 7-3, enquanto que a Tabela 7-4 apresenta os custos das soluções para os mesmos casos de teste, considerando as restrições de adjacência. Em cada uma das duas tabelas é indicado, para cada caso de teste, o melhor custo, o pior custo e o custo médio do

conjunto dos dez resultados obtidos, bem como os respectivos números de gerações e tempos de processamento gastos.

Tabela 7-3: Resultados para os cinco casos de teste sem a utilização de restrições de adjacência.

Problema		**Custo**	**Nº Médio de Gerações**	**Tempo de Processamento**
ppli8	Melhor	24239	50	85
	Pior	28799	40	69
	Média	26253	45	77
ppli10	Melhor	21653	110	611
	Pior	24314	60	307
	Média	22625	73	406
ppli10c	Melhor	23161	60	644
	Pior	25407	40	588
	Média	24286	49	609
ppli15	Melhor	25270	140	3797
	Pior	27560	50	1163
	Média	26802	88	2188
ppli24	Melhor	96232	130	1352
	Pior	109505	110	1168
	Média	103398	131	1329

Comparando os resultados da Tabela 7-3 com os da Tabela 5-9. que contém o custo das melhores soluções obtidas com o *LaRLo* ao fim de uma hora, verifica-se que o *LayGeRL* proporciona sempre soluções melhores. Também se verifica que necessita normalmente de um período de processamento inferior a uma hora. (registe-se a excepção relativa à melhor solução obtida para o problema 'ppli15' que necessitou de um período de processamento ligeiramente superior a uma hora).

Outra excepção foi que, para o problema 'ppli24', a pior solução possui um custo marginalmente superior ao custo da solução obtida com o *LaRLo*. Esta situação pode estar relacionada com o facto de que os valores escolhidos para os quatro parâmetros do AG não corresponderem à gama de valores de que se esperariam os melhores resultados. Estes valores foram escolhidos apenas com objectivo de a resolução do problema não demorar mais do que uma ou duas horas. De qualquer forma, em termos médios, mesmo com estes valores escolhidos para os parâmetros do AG, a qualidade das soluções é melhor.

Tabela 7-4: Resultados para os cinco casos de teste com a utilização de restrições de adjacência.

Problema		**Custo**	**Nº Médio de Gerações**	**Tempo de Processamento**
ppli8	Melhor	22559	20	45
	Pior	23519	30	83
	Média	22943	27	73
ppli10	Melhor	18734	70	1070
	Pior	20977	30	617
	Média	20255	54	921
ppli10c	Melhor	18925	100	2576
	Pior	21754	40	1917
	Média	20652	52	1736
ppli15	Melhor	25745	80	9033
	Pior	27303	110	17759
	Média	26622	99	12991
ppli24	Melhor	94911	370	6465
	Pior	101326	230	6042
	Média	97268	337	6869

Considerando agora a Tabela 7-4 e comparando-a com a Tabela 7-3, verifica-se que a qualidade das soluções é geralmente melhor quando se colocam restrições de adjacência. No entanto, o tempo de processamento necessário tem tendência a crescer com o aumento da dimensão do problema. Registe-se que para o caso de teste 'ppli15' o tempo médio de processamento é aproximadamente seis vezes superior e para o caso de teste 'ppli24' o tempo médio de processamento é aproximadamente cinco vezes superior, mesmo com um tamanho da população inferior em 25%.

Comparando os resultados da Tabela 7-4 com a qualidade das soluções obtidas com o *LaRLo*, observa-se também soluções de superior qualidade quando se utiliza o *LayGeRL* e tendo em conta as restrições de adjacência.

Como conclusão final, a qualidade das soluções obtidas com o *LayGeRL* são geralmente melhores do que as que resultam da utilização do *LaRLo* para as mesmas condições (sem ou com restrições de adjacência). A Tabela 7-5 e Tabela 7-6 são a prova deste facto, ao mostrarem os custos das melhores soluções obtidas, sem e com restrições de adjacência respectivamente, com o *LaRLo* e com o *LayGeRL* para o cinco casos de teste considerados.

Tabela 7-5: Comparação do *LaRLo* com o *LayGeRL* sem a colocação de restrições de adjacência.

Problema		**ppli8**	**ppli10**	**ppli10c**	**ppli15**	**ppli24**
LaRLo	Custo	31377	25836	25926	29286	109372
	λ	Menor	Médio	Menor	Menor	Médio
	PE	*PE*5	*PE*5	*PE*5	*PE*5	*PE*1
LayGeRL	Custo	24239	21653	23161	25270	96232
	Tempo (segundos)	85	611	644	3797	1352

Recorde-se que, no *LaRLo*, a pesquisa das soluções é interrompida ao fim de um determinado período de tempo (uma hora). Além disso, verifica-se também que o tempo de processamento necessário para encontrar as soluções é geralmente bastante

menor com o *LayGeRL* do que o necessário com o *LaRLo*. Há ainda a considerar que, para o caso do *LaRLo*, as melhores soluções obtidas para os cinco casos de teste resultaram da utilização de diferentes heurísticas da ordem de selecção de variáveis diferentes (λ) e procedimentos de etiquetagem (PE).

Tabela 7-6: Comparação do *LaRLo* com o *LayGeRL* utilizando de restrições de adjacência.

Problema		**ppli8**	**ppli10**	**ppli10c**	**ppli15**	**ppli24**
LaRLo	Custo	22784	23422	22985	29361	114401
	λ	Partição	Partição	Menor	Médio	Menor
LayGeRL	Custo	22559	18734	18925	25745	94911
	Tempo (segundos)	45	1070	2576	9033	6465