# Tabu Search for target-radar assignment

by

## Magnus Hindsberger and René Victor Valqui Vidal

Department of Mathematical Modelling (IMM)
Technical University of Denmark
DK-2800 Kgs. Lyngby, Denmark
E-mail: mh@imm.dtu.dk ; vvv@imm.dtu.dk

**Abstract:** In this paper the problem of assigning air-defense illumination radars to enemy targets is presented. A Tabu Search metaheuristic solution is described and the results achieved are compared to those of other heuristic approaches. Implementation and experimental aspects are discussed. It is argued that Tabu Search could be used in near realtime decision making systems.

**Keywords:** combinatorial optimization, metaheuristics, Tabu Search, assignment.

## 1. Introduction

In case of an enemy air raid an air defense system should engage enemy fighters, helicopters etc. with the available missiles so that the most threatening and valuable enemy targets are engaged first. Since the air picture is changing constantly, the assignment of the missiles to the wanted targets must be continuously updated. Unfortunately, the updating is non-trivial because even for relative small problems, the missiles, the controlling radars, and the targets can be combined in a huge number of ways during a time period.

In addition, the optimal solution does not necessarily have to include all the targets and the targets included must be assigned during specific time windows as they enter or exit the effective range of the system.

The complex nature of this assignment problem and the fact that a solution must be found within a strict time span in order to keep the assignment up-to-date with the air picture, makes it a very interesting combinatorial optimization problem.

Such a problem is approached using heuristic principles (see Silver et al., 1980) due to its complexity that makes any global approach unsuitable. Moreover, metaheuristic approaches such as Local Search, Tabu Search, Simulated

(see further Glover and Laguna, 1993, Vidal, 1993, and Pirlot, 1992). Note that the approach used in this paper was tried for research purpose only. The algorithm adopted by the Danish airforce is, although also heuristic, very different from the one presented here.

In Section 2, an outline of the system in study is presented. In Section 3, the target-radar assignment problem is formulated and modelled. A two-step heuristic approach is described in Section 4, while Section 5 presents the first step based on Tabu Search, and Section 6 presents the second step based on Steepest Ascent. Section 7 presents our numerical results obtained in a series of experiments. Section 8 presents improvements in the developed procedure. Section 9 gives some results of experimentations on the problem of quality vs. time usage. Finally, the last section presents our final conclusions.

## 2. The DEHAWK system

DEHAWK is a modification program of the Danish HAWK air-defense missile units (for details, see Cullen and Foss, 1998). This paper will discuss a simplified DEHAWK system looking only at two types of units: the illumination radars (denoted HIPIR's, HIgh-Power Illumination Radar) and the HAWK missiles. Note that the names HAWK and DEHAWK are used interchangeably and both refer to the updated system.

A DEHAWK battery will usually have two HIPIR's, while a batalion will control up to four batteries and thus 8 HIPIR's.

The HAWK system is classified as a medium range air defence system. If a target is to be engaged by a HAWK missile, the target has to be illuminated by one of the HIPIR radars. This means that the HIPIR starts tracking the target by continuously keeping it the target of the radar emission. Some of the emission is reflected from the target and is picked up by the HAWK missile, which uses it to keep the guidance towards the target. The process is sketched in Fig. 1.

In general, air defense units are made up of *firing units*, which will be denoted FU's. A FU is the smallest unit which can independently engage a target. Since the guidance method limits a HAWK squadron to only have two HAWK engagements running at the same time (i.e. one for each HIPIR), the HIPIR's can be seen as the FU's.

An *engagement* will denote the time sequence of assigning a single target, firing at it, until it has been determined whether or not the target was hit. The time period where it is possible for one of the FU's to engage the target is denoted the *engagement area*.

The attacking side sending the air raid will be known as the orange side, while the defending side, using the HAWK system, is the blue side.

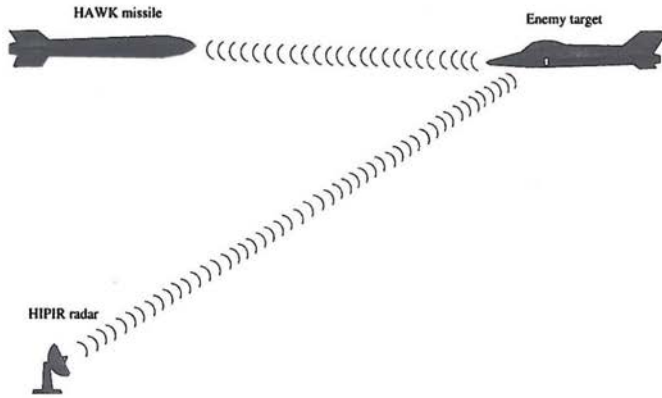When engaging, you have a certain probability of shooting down the target

Figure 1. The HAWK guidance system

present paper be denoted *Pkill*. At the time the engagement ends, a successful outcome will mean a benefit for the blue side. This benefit, denoted the *military value* in this paper, must be specified by the user. It usually depends upon the threat it constitutes to the blue at the specific time.

## 3. Problem formulation

The target-radar assignment problem can be formulated as: In a specific time period, called the *planning period*, to decide which targets to be engaged, at which time, and which HIPIR to be used for controlling the missiles flight. The length of the planning period can vary, but it is usually chosen to be between 2 and 5 minutes.

The assignment has to be updated continuously according to changes in the target course and speed. Therefore, the procedure for solving the described problem must calculate a new assignment plan every time the search radar has finished a rotation. During a single rotation the information about all targets in the airspace is updated. Since a rotation only takes about 3 seconds, the time span, within which the assignment plan should be calculated, is very strict.

For the sake of simplicity the missile launcher assignment part has been omitted. In addition, the functions for calculating *Pkill*, the missile flighttimes, and the military values for targets have been simplified.

In order to evaluate whether a procedure can be accepted, some criteria must be specified. Some of the criteria are *strong*, meaning that they should always be fulfilled.

For this problem the strong criteria are:

- Speed: The procedure must return a solution within the 3-second time

- Maximizing: The expected value of the objective function should be within 5% of the global optimum.

In this case the speed criterion is actually implemented as a constraint. A set of weak criteria can be defined too. They describe the desirable but not demanded properties of the procedure. The following weak criteria have been defined for this problem:

- Range: The number of targets that come within their own firing range of a blue target before being engaged, should be minimized.
- Robustness: The algorithm should, when run in real-time, not change the recommended solution for every new 3-second run unless targets are maneuvering or new targets appear.

The first of the weak criteria can possibly be chained to the strong criterion of maximization of the objective value by a suitable definition of the objective function. The robustness criterion is included because the users of the system would loose confidence in a system that for every 3-second period makes changes in its recommended assignment plan.

For the sake of comprehensiveness a mathematical model of the problem, though not complete, is formulated below.

In the model the following indices are used:

$i = 1, 2, \ldots, n$ denotes a target from the list of targets.

$j = 1, 2, \ldots, m$ denotes the HIPIR used to illuminate the target.

$k = 1, 2, \ldots, o$ denotes the specific missile used for an engagement.

The decision variables to be determined are:

$t_{ijk} \in \mathbb{R}$ denotes the time where the firebutton is pressed for missile $k$ to engage target $i$ controlled by HIPIR $j$.

$x_{ijk} \in \{0, 1\}$ is equal to 1 iff the engagement including target $i$, HIPIR $j$, and missile $k$ is included in the assignment plan for the planned period.

The following functions are used in the model:

$mv(t_{ijk})$ calculates the military value of target $i$ if engaged by missile $k$ at the time $t_{ijk}$.

$Pk(t_{ijk})$ calculates the *Pkill* of target $i$ if engaged by missile $k$ at $t_{ijk}$.

$f(t_{ijk})$ returns the time of the intercept of target $i$ using missile $k$ launched at $t_{ijk}$ and controlled by HIPIR $j$.

In addition, the following parameters are defined:

$eng\_start_{ij}$ is the first time when the firebutton can be pressed to engage target $i$ — i.e. $i$ enters the intercept area of HIPIR $j$.

$eng\_end_{ij}$ denotes the time when target $i$ exits the intercept area of HIPIR $j$.

$r\_hipir_j$ is the time when HIPIR $j$ are ready for the first engagement.

$t\_lk$ is the reaction time between the instance when the decision to shoot is taken and when the firebutton can be pressed due to the radar lock-on

$t\_ev$ is the time between the instance when a target is hit and when it has
    been evaluated whether it was shot down. If not — it probably should be
    reengaged.

For every target, *Pkill* is the probability of shooting it down at a certain
time and the military value is the benefit of the kill if made. The total expected
benefit of an assignment plan can be calculated as follows:

$$maximize \quad \sum_{ijk}(Pk(t_{ijk}) \cdot mv(t_{ijk}))x_{ijk} \tag{1}$$

Restrictions should ensure:

1. that the targets are engaged within valid intercept areas and that the
   reaction time will not let a target get out of range before the actual launch.
2. that a single HIPIR is not used in more that one engagement simultane-
   ously.
3. that one target is only assigned once in the assignment plan.

Note that in this implementation a target is only engaged with a single
missile. For some very dangerous targets it might be better to assign several
HIPIR's to them in order to engage the targets with multiple missiles. See
Hindsberger (1998-I) for a more comprehensive description on the military value,
the *pkill* function, engagement area, etc.

The number of HIPIR's, corresponding to the index $m$, can be up to 8
and it is not unlikely to expect more than 30 targets (index $n$), of which all
should come within the engagement area of a HIPIR during the planning period.
Many of the combinations are prohibited in the cases where some targets cannot
be engaged within their valid engagement areas, that is not known before the
specific combinations have been checked. So, the number of combinations in the
problem is huge, taking into account that the time limit is 3 seconds.

## 4. A heuristic approach

The target-radar assignment problem is a non-standard non-linear mixed integer
problem and due to its complexity is even for small dimensions rather difficult
to solve using an exact approach.

The complexity of the problem led to a two-steps heuristic for solving the
problem. In the first step the assignment of targets to the HIPIR's is made.
That is, it is decided which target should be illuminated by which HIPIR, and
the succession in which each HIPIR should illuminate the targets. The second
step optimizes the start time of the engagements assigned to a HIPIR, making
the engagements start at the most desirable time. Throughout this paper the
engagement start time denotes the time, when the firebutton is pushed in order
to launch a missile for an engagement.

For the first step, which is the integer part of the problem, it was decided to
solve it using general search heuristics, or more specifically the neighbourhood-

order to get an insight into the problem, the simplest of these heuristics — the Steepest Ascent (greedy) heuristic was implemented. From this, one can get useful information about the behaviour of the problem and some objective function values, which one can compare with the results of the more advanced heuristics.

Of the more advanced neighbourhood-based heuristics, Multistart Ascent, Simulated Annealing and Tabu Search were tried, and since all could be based on the neighbourhood function already written for the Steepest Ascent algorithm, implementations were quickly done.

For the second step only a continuous version of the Steepest Ascent heuristic was implemented.

The implementation of the two steps including the neighbourhood structures used and Tabu Search memory implementation is described below.

## 5. Step 1 — Assignment optimization

In Fig. 2 a pseudo-code of the classic Tabu Search program is shown. The neighbourhood implementation is described below. The *Time_opt* procedure is the second step carrying out the engagement time optimization and is covered in Section 6.

```
begin
  initialization
  curr_x := make_init_solution
  best_x := curr_x
  best_obj := Time_opt(best_x)
  unchanged := 0
  k := 0
  while (unchanged < max_unchanged and k < max_total) do
    Y := find_neighbourhood(curr_x)
    (curr_x,curr_obj,tabu) := find_best(Time_opt(Y), tabu)
    if best_obj < curr_obj then
      best_x := curr_x
      best_obj := curr_obj
      unchanged := 0
    else
      unchanged := unchanged + 1
    endif
    k := k + 1
  end
end
```

The function, *find_best*, which evaluates the generated neighbours takes the tabu memory as parameter, returning the best of the neighbours and the corresponding objective function value, $C(x)$, depending on the current recency and frequency memory. Then, the current solution is appropriately set and the tabu memory is updated according to this choice. The best value obtained at this point is updated if the value of $C(x)$ of the new solution is an improvement.

A datastructure for storing the assignment of both the current solution and the solutions of the generated neighbours is defined. The basis is the *assignment-*class, which contains all the information about a single assignment, identified with the ID-number of the assigned target. An array of assignments is defined for each HIPIR-radar, containing the assignments of that particular HIPIR. An additional array is defined containing all the non-assigned targets. A sample assignment plan is shown in Fig. 3. The system shown consists of four HIPIR's and seven targets. Three of the targets (1, 2 and 5) are unassigned, while target 3 has been assigned to HIPIR 1 and HIPIR 3 should engage target 4, 6 and 7 in this succession.

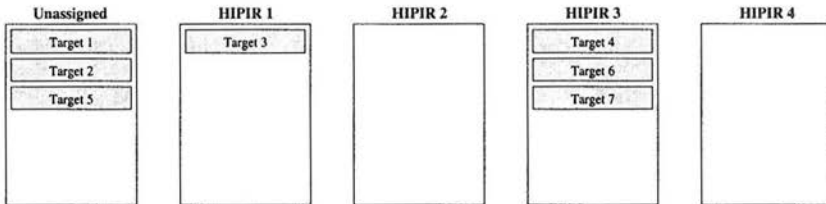| Unassigned | HIPIR 1 | HIPIR 2 | HIPIR 3 | HIPIR 4 |
|---|---|---|---|---|
| Target 1 | Target 3 | | Target 4 | |
| Target 2 | | | Target 6 | |
| Target 5 | | | Target 7 | |

Figure 3. The data-structures where a solution is stored

The initial assignment was chosen always to be the zero-assignment, i.e. the assignment where all targets are unassigned, and thus stored in the unassigned array. The stochastic nature of the neighbour generating function, as shown later, will make sure that every possible assignment plan can be reached from this assignment. So, there was no reason to start with some or all of the targets randomly assigned to the HIPIR's.

The search for a new assignment is done within the while-loop of Fig. 2. At first a part of the neighbours to the current solution is found. This is done by the neighbour generation function using the neighbourhood definition described below.

## 5.1.  The neighbour generating function

For the implementation it was decided to create a function, which generates

can be user-specified, but has an upper limit depending upon the number of HIPIR's and targets.

A neighbour is defined as the move of one target and not as an exchange of targets. If $n$ neighbours are to be generated, only the first one will be randomly chosen, while the $n - 1$ next neighbours are defined by a succession, which will be described later.

When picking the first neighbour a random HIPIR (with at least one target assigned to it) to move from is selected. From the array of such HIPIR's a random target to move is drawn. Note that all HIPIR's have an ID number, where 0 denotes the array containing the unassigned targets. If moving from HIPIR 0 (the array of unassigned targets) the HIPIR to move to is randomly selected from among those with an ID higher than 0. Else the algorithm just picks a random one of all the arrays to decide where to move. If the HIPIR moving from is the same as the one moving to, the neighbours generated just change in the succession in which HIPIR engages the targets. When the two HIPIRs' ID numbers are different, the neighbours generated in this case are the possible placements of the moving target in the succession of targets assigned to the receiving HIPIR.

The following steps define the succession of the neighbours, once the first one has been selected. The steps are taken until the wanted number of neighbours has been generated. Because of the number of special cases, it might look complicated, but the example below should help to clear out any confusion.

- The selected target is moved to the HIPIR with the next higher ID number, starting over from 0, when the highest number has been reached. If the target is moved from HIPIR 0 (unassigned), the unassigned number is excluded.

- If it has been tried to move the target to all possible HIPIR's, the procedure switches to the next target (in the succession) of the HIPIR's moving from. If the last target of that HIPIR has been reached, the HIPIR to move from is switched to the next higher number, again starting over from 0 if the highest ID number has been reached.

Note that each time the HIPIR's or target ID is changed, the number of new neighbours, which can be created, will vary depending upon the current solution.

EXAMPLE 5.1 (THE TABU SEARCH NEIGHBOURHOOD) *Consider the situation where the solution pictured in Fig. 3 is the current solution and 9 neighbours are to be generated. The first neighbour is chosen in the following way:*

- *One of the nonempty HIPIR arrays is chosen to move from — in this case HIPIR 3.*

- *Of the three targets assigned to HIPIR 3, the second target is randomly selected, i.e. the target with ID 6 is to be moved.*

- *Now a HIPIR to move to is chosen. In this case any would do, and*

*This gives neighbour 1 pictured in Fig. 4. If the steps described above are used to pick the next $n-1$ neighbours, the neighbours numbered from 2 to 9 in the figure are generated. In neighbour 1 target 6 is moved to HIPIR 2. Since the array of HIPIR 2 is empty, this can only be done in one way. So, when generating neighbour 2, the target now has to be moved to the next HIPIR array, which in this case is HIPIR 3 — the one moving from. Since there are more targets assigned to HIPIR 3 than target 6, it can be put in other places in the succession of engagements resulting in neighbours 2 and 3. When coming to the array of unassigned targets, there is no defined succession of those targets. So target 6 is just added to the array making neighbour 5.*
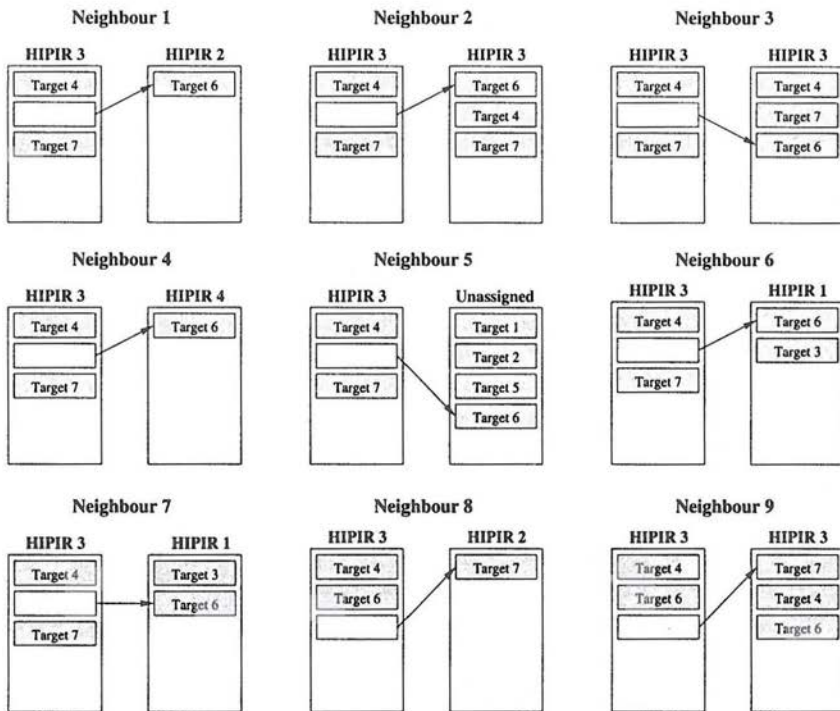


Figure 4. Neighbours generated in Tabu Search

*Before all 9 neighbours have been generated, target 6 has been moved to all possible positions. So, the algorithm switches to the next target in the succession, which is the next target — if any — assigned to the same HIPIR. In this example it is target 7. So, for the neighbours 8 and 9 it is target 7 to be moved.*

*If even more neighbours were generated, target 7 would at a point have been*

*be used for generating the next neighbours. In this example it is target 1 of the
unassigned array, since there are no more targets assigned to HIPIR 3 and no
one is assigned to HIPIR 4.*

With this neighbourhood definition a neighbour is a small change to the
existing solution and it assures that all possible solutions can be reached from
any other solution, which are some of the basic properties a neighbourhood
should possess.

### 5.2.  Implementing memory

As the tabu element either just the moving target or a combination of the
receiving HIPIR and the moving target can be used. Thus, for instance, if
target 6 just has been moved from HIPIR 3 to HIPIR 2, either the moving of
target 6 (no matter where-to) or the moving of target 6 (no matter from which
HIPIR) to HIPIR 2 is declared as tabu for the next *tabulength* iterations. In this
implementation the tabu length was fixed for the duration of a computation run.

Both of the above mentioned tabu structures have been implemented in order
to see any differences in performance. The frequency memory is implemented
too and has to fit the tabu structure used. So, in the first case the penalty
added to $C(x)$ is a certain percentage of the number of times the target has
been moved, while in the other case — some percentage of the times the target
has been moved to that specific HIPIR. The percentage is user-specified in both
implementations.

EXAMPLE 5.2 (THE TABU SEARCH FREQUENCY MEMORY) *The frequency
memory is used to diverge the search into new areas. In this case it is imple-
mented as a penalty to the objective function. The penalty added to the objective
function is a user specified percentage of the number of times the move to the
investigated neighbour already has been taken. If 10% has been specified and the
number of times target 6 has been moved is 10, $C(x)$ of the neighbour describing
this move will be lowered by 1. If target 6 had been moved 100 times, the penalty
would be 10 instead, making the less taken moves more interesting.*

In addition to the frequency penalty, the tabu length, the number of neigh-
bours generated, and the parameters of the stopping criterion (fixed time or
fixed number of iterations) are user-specified, making it possible and easy to
test the dependence of those parameters.

### 5.3.  The objective function

When the best of the neighbours is to be found, the change in the objective
function value $C(x)$ that each neighbour results in, is calculated. $C(x)$ was
defined as the sum of the military value of a target times the probability of

time the missile hits (or misses) the target, you have to find the *optimal time* for each engagement to start. This is done by the *Time_opt* procedure described below, where the *optimal time* also will be defined.

## 6.  Step 2 — Time optimization

In order to find the optimal start times of the engagements of a HIPIR, this procedure is used. Using a nontrivial neighbourhood, it features the Steepest Ascent method, since all possible neighbours are checked at each iteration. A pseudo-code of the procedure can be seen in Fig. 5.

```
procedure Time optimization
check_if_possible
if no_of_targets = 1 then
   update(assign)
else
   begin
     initialization
     change := true
     while change do
       change := false
       for i := 1 to no_of_targets do
         imp1 := move_forward_check(i)
         imp2 := move_backward_check(i)
         improvement := max(imp1,imp2)
         if improvement > 0 then
            update(assign)
            change := true
         endif
       end
     end
   end
```

Figure 5. Pseudo-code of the time optimization procedure

When the procedure is called, it has to make sure it is possible within the allowed planning time to engage all the assigned targets. The *check_if_possible* procedure will do this as well as put the assignments in a tight succession, which is the succession, where each engagement starts as soon as the previous engagement is finished. This solution is the initial solution of the algorithm, but since it is not necessarily the optimal time assignment, it has to be tested whether some targets could be engaged at more appropriate times, improving

## 6.1.   Optimal time — critical line

The *optimal time* for an engagement refers normally to the time where an engagement should be started, in order to have the highest probability of shooting down the target. An exception is, if a target has crossed the critical line before it is hit by a missile fired at the optimal time using the definition above. In this case the optimal time, instead, denotes the engagement start time to be used, so the target is hit as it reaches the critical line. This is done in order to make the algorithm engage the targets before they cross the critical line, if possible, which was one of the properties the algorithm should fulfil.

## 6.2.   Neighbourhood generation

When only one target is assigned to a HIPIR, it is easy. A precalculated table contains the optimal start time of an engagement for each target-radar combination. So, this start time is used, unless it is later than the planning time allows. In this case the target is engaged at the latest possible time, instead.

With more than one target assigned, the neighbourhood generation becomes more complicated, since the engagements can block each other, so one or more of them cannot be engaged at the optimal time. Three points of time are defined for each target. The earliest possible, *EST*, the optimal, *OST*, and the latest possible engagement start time, *LST*. The first and the last of those can change as the engagement start time of either the previous or the next engagement is changed, but they will never cross the boundaries of the possible intercept period. It should be noted that the optimal engagement start time is not necessarily between the other two, as seen for the third engagement of Fig. 6. In this figure engagements 2 and 3 block each other.
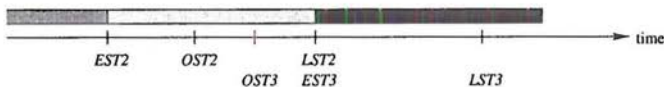


Figure 6. A sequence of engagements where a blocking occurs, since the EST of the last engagement is later than the OST

When such a blocking occurs between two engagements, a boolean variable will be set for use in the moves explained below.

The main part of the *time_opt* procedure is the *for*-loop, which for every assigned target will calculate the change in $C(x)$ for each of the possible, following

**Forward move:** The engagement is moved forward (i.e. delayed), if the engagement is not blocked by the next and either the engagement start time is earlier than the optimal of that target-radar combination or it is blocking for the previous engagement to move forward.

**Backward move:** The engagement is moved backwards (i.e. started earlier), if the engagement is not blocked by the previous and either the current start time is later than the optimal of that target-radar combination or it is blocking for the next engagement to move backward.

The most promising "move" can now be found, as the one with the greatest positive change to $C(x)$. Since all possible neighbours are tested, this is a Steepest Ascent algorithm. So, until no improvement can be found for any of the targets, the loop above is repeated.

This neighbourhood definition should both make a neighbour a small change compared to the existing solution and ensure that all "optimal" solutions can be reached within a finite number of iteration from any other solution. The notation "optimal" is used, since not all possible time displacements are defined in this implementation. But the ones defined should, with the assumption described below, include the optimal solution.

The assumption that has been made is that the increase or decrease rate of the benefit value of each target (i.e. the *Pkill* times the military value) is constant between $EST$, $OST$, and $LST$, as shown in Fig. 7.
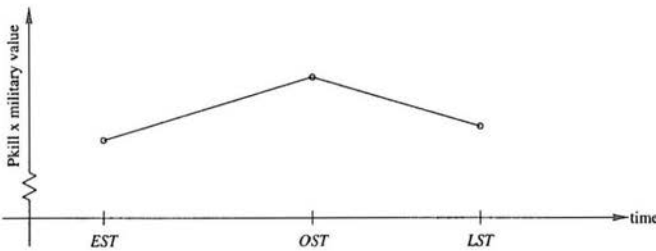


Figure 7. For the implementation it has been assumed that the increase or decrease rates between the points are constant as shown in the graph

With the mentioned assumption the optimal solution is still contained in the solution space. This can be seen in the following way. If no blockings occur, the optimal solution is included, since all $OST$ points are included. If a blocking occurs as the one in Fig. 6, both of the start times are tried to be moved $\Delta T$ forward as shown in Fig. 8. If the gain in benefit of engagement 2 is greater than the loss of benefit of engagement 3, $\Delta T$ should be as great as possible, which is the greater of the intervals $[EST2, LST2]$ and $[EST3, LST3]$. If the

be any of the two values. Since all the *EST* and *LST* points are included in the solution space, the optimal solution will also be contained in it, because of the assumption of constant increase/decrease rates.
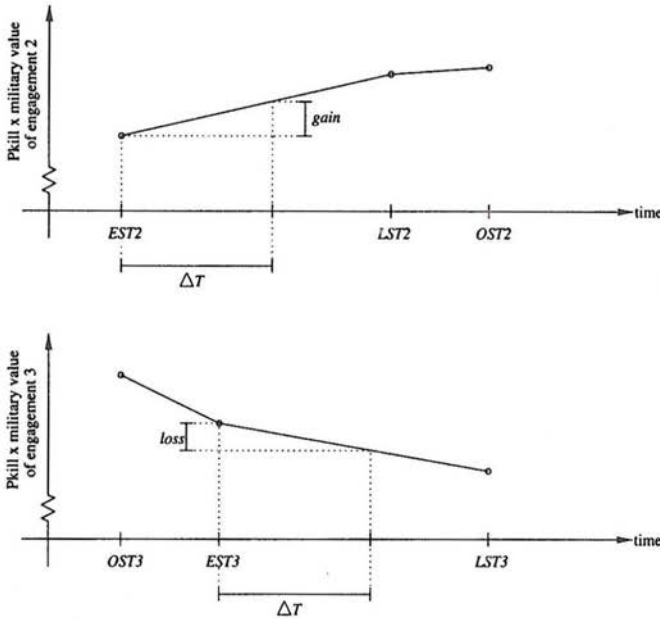


Figure 8. The upper plot shows the benefit of engagement 2 of Fig. 6 while the lower is the benefit of engagement 3

During the testing this procedure turned out to be very good in finding the global optimal solution since only in rare occasions an assignment blocked another one, which could have improved the solution.

After the time optimization the partial objective function value of the used HIPIR is stored in memory and used by the *find_best* procedure to find the best of the generated neighbours.

## 7.   Experiments

The stochastic nature of the algorithms make them in general return different solutions to the same problem. In order to find the expected value of $C(x)$ of the algorithms for a given scenario, an automated program was written. It ran the algorithm a user-specified number of times with the same parameters, except for the random number seed, calculating the sample mean value and variance

The results shown in this paper are, unless otherwise stated, a sample mean value of a 100 runs batch. All the computations were done on a Digital VAX 4000-60 workstation.

It was initially assumed that a 100 times improvement in the computation time could be gained, if the algorithms were reprogrammed in C++ and run on a modern workstation. Therefore, the algorithms were allowed to run for 300 seconds in the tests, and were still assumed to be within the time limit of 3 seconds. Whether or not this assumption holds is discussed in Section 9.

## 7.1.   Test scenarios

The algorithms were tested on various scenarios, which had a different number of targets and HIPIR's. A scenario with two HIPIR's and seven targets will in this paper be denoted as a 2-7 scenario, while a scenario with 8 HIPIR's and 13 targets is thus denoted an 8-13 scenario. The density of a scenario describes the number of targets compared to the number of HIPIR's. A 2-13 scenario is considered dense while an 8-7 scenario is sparse. The size of a scenario depends both on the number of HIPIR's and the number of targets. A scenario will in most cases be considered small, if it has less than ten targets or four HIPIR's.

In all scenarios the HIPIR's were located in pairs and the targets incoming were all jet fighters. The planning time was set to 200 seconds.

For none of the scenarios the global optimal value of $C(x)$ is known for sure. So, the mean values obtained will be compared with the best value of $C(x)$ found for that particular scenario, which should be very close to, if not the global optimum. For the 2-13 and the 8-13 scenarios, which will be used most widely, the best values found were 247.73 and 482.67, respectively, and the "optimal" value will in this paper, when mentioned, refer to one of these values.

## 7.2.   Tabu Search results

As described in Section 5, two versions of Tabu Search were implemented with the tabu definition being the difference. The one denoted TS-simple uses the "simple" tabu definition with only having the moving target as the tabu element, while the other, TS-advanced, uses the more advanced definition having the tabu element as a combination of the moving target and the receiving HIPIR.

For the first experiments the stopping criterion was to stop after $uc$ unchanged iterations. Initial testing showed that $uc = 80$ gave the best results compared to the time used for most scenarios. The testing is described in more detail in Hindsberger (1998-I). Experiments to assess the influence on the two algorithms from the number of neighbours generated and the length of the tabu list (i.e. the *tabulength*) were made then, keeping the frequency penalty and the

When using the simple tabu definition a 10% frequency penalty was used, since only 13 tabu elements exist, being the 13 targets. This value was a guess but from the earlier results achieved by the other methods, neighbours often differed less than 0.5 in their values of $C(x)$, so the 10% penalty should be sufficient to diverge the search to other areas, while still being small enough not to disrupt the basic search with "noise".

When using the advanced definition the number of tabu elements was $8 \times 13$, so the penalty was for this definition increased to 100% for similar reasons as above.

With all other parametres fixed, the frequency penalty was varied, to see if the values used until now were actually optimal. While TS-simple turned out to do better using a penalty of 15% rather than the previously used 10%, the 100% penalty used by TS-advanced looked optimal. The results when using TS-simple on the 8-13 scenario can be read from the graph in Fig. 9. The dashed line is the best fit by a polynomial of degree 2.
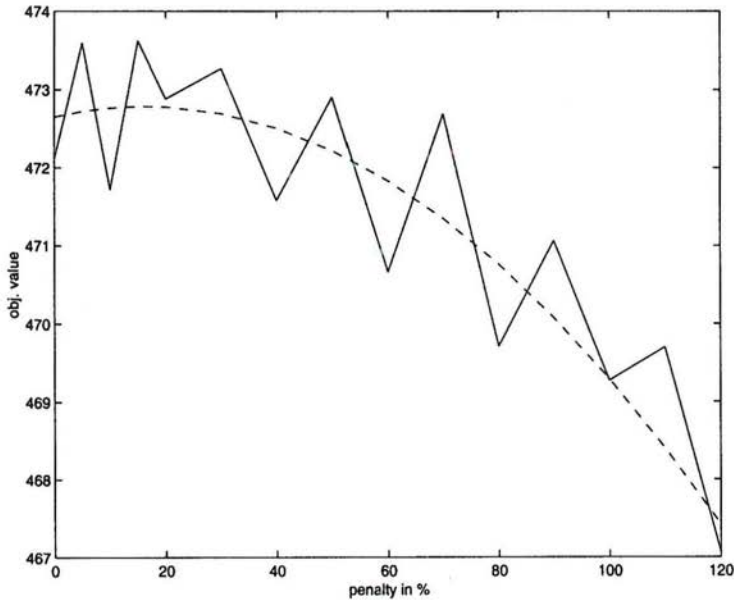


Figure 9. The dependence on the penalty-% for TS-simple using the 8-13 scenario

Table 2 shows the best values of the tabulength (denoted $T$), the number of neighbours generated at each iteration ($NB$), and the frequency penalty (% penalty) for the two different versions of TS used on the 2-13 and the 8-13

| Scenario | ID | mean | s. dev. | max | min | % best |
|----------|-----|--------|---------|--------|--------|--------|
| 2-13 | 1 | 227.14 | 8.78 | 247.56 | 205.42 | 8.31 |
|      | 2 | 229.01 | 9.04 | 247.73 | 208.83 | 7.56 |
| 8-13 | 3 | 473.62 | 6.86 | 482.67 | 449.89 | 1.87 |
|      | 4 | 472.77 | 6.91 | 482.03 | 438.58 | 2.05 |

Table 1. Statistics on some of the best TS runs. The ID numbers refer to the parameter values shown in Table 2 below

| ID | Scenario | method | NB | T | % penalty |
|----|----------|-------------|----|----|-----------|
| 1 | 2-13 | TS-simple | 16 | 8 | 15 |
| 2 | 2-13 | TS-advanced | 16 | 12 | 100 |
| 3 | 8-13 | TS-simple | 34 | 4 | 15 |
| 4 | 8-13 | TS-advanced | 34 | 12 | 100 |

Table 2. Table showing the parameters used in Table 1. $NB$ denotes the number of neighbours generated, $T$ denotes the tabu list length and the % penalty, the penalty to the frequency memory

Looking at the Tables 1 and 2 one can see that the actual performance of TS-simple is similar to that of TS-advanced. The optimal numbers of neighbours to be generated might look high, but experiments showed that with a lower number of neighbours generated — where more iterations can be performed instead — the algorithms come out with inferior solutions.

## 7.3.  Comparison with other methods

A total of 5 different algorithms were implemented: A Local Search algorithm (LS-Single), performing *one* search, the same algorithm performing searches for the time available (LS-Multi), a Simulated Annealing algorithm (SA), as well as TS-Simple, and TS-Advanced. For a thorough presentation of these approaches see Hindsberger (1998-I). For a fair comparison of the algorithms, they were run for a fixed 300 CPU-seconds period (of which LS-Single only used a few seconds). The algorithms were tried using both the 2-13 and the 8-13 scenarios as before, in order to see any dependence of the scenario density. Note that the results in this section are based on 50 runs of each and not the 100 as used by the test program earlier in this section.

The parameters used are the same as the ones found to work best below 300 seconds in average computation time. For the two Tabu Search algorithms they are the ones shown in Table 2. When now running for a fixed period, the individual runs, which took longer without the time limit, should return inferior solutions, while the ones finished before the 300 seconds might obtain better results. As it can be seen later in Tables 3 and 4, the mean value of $C(x)$

| Method | mean | s. dev. | max | min | % best |
|--------|------|---------|-----|-----|--------|
| LS-single | 195.46 | 20.12 | 241.63 | 155.02 | 21.10 |
| LS-multi | 230.40 | 6.45 | 245.44 | 218.87 | 7.00 |
| SA | 218.18 | 13.45 | 246.30 | 185.93 | 11.93 |
| TS-simple | 233.31 | 6.93 | 247.56 | 221.97 | 5.82 |
| TS-advanced | 235.08 | 6.53 | 247.73 | 222.14 | 5.11 |

Table 3. Descriptive statistics of the different algorithms when applied to the 2-13 scenario

| Method | mean | s. dev. | max | min | % best |
|--------|------|---------|-----|-----|--------|
| LS-single | 453.84 | 16.09 | 478.52 | 419.00 | 5.97 |
| LS-multi | 473.51 | 3.89 | 482.62 | 464.21 | 1.90 |
| SA | 465.92 | 8.72 | 480.00 | 440.10 | 3.47 |
| TS-simple | 475.21 | 4.56 | 482.67 | 459.66 | 1.55 |
| TS-advanced | 472.41 | 5.27 | 480.81 | 457.36 | 2.13 |

Table 4. Descriptive statistics of the different algorithms when applied to the 8-13 scenario
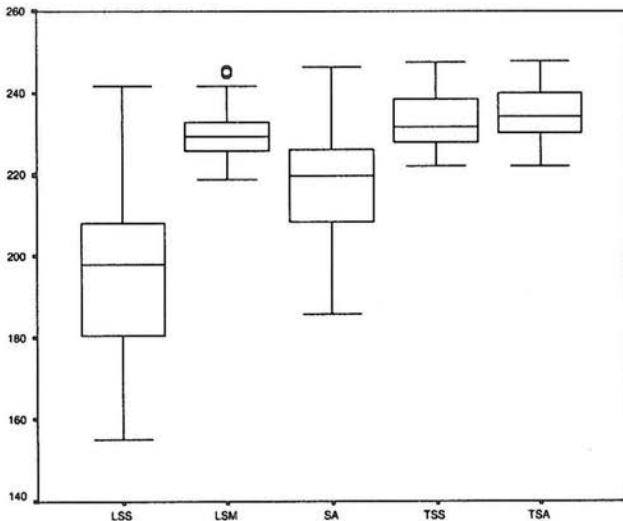


Figure 10. Boxplot showing the performance of the different algorithms applied
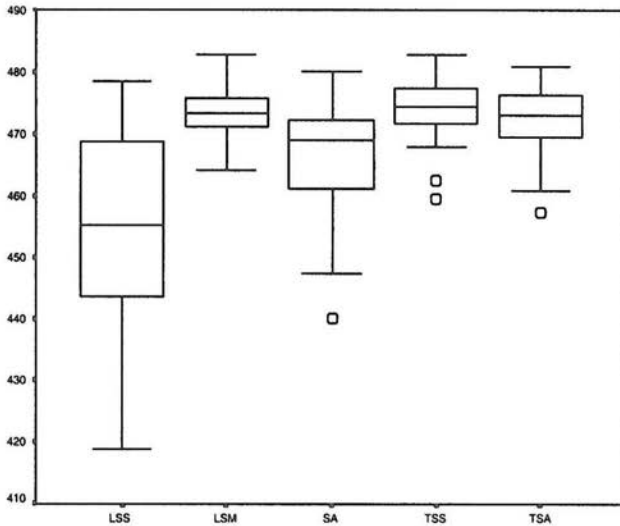
Figure 11. Boxplot showing the performance of the different algorithms applied to the 8-13 scenario with parameters as described in Table 2

Tables of the descriptive statistics of $C(x)$ obtained using the different methods are shown in Tables 3 and 4, while boxplots showing the distributions for comparison can be found in Figs. 10 and 11. LSS and LSM denote the LS-Single and LS-Multi methods, while TSS and TSA denote the TS-Simple respectively TS-Advanced methods.

From the tables and figures it is concluded that TS-advanced is the best suited algorithm of all since it is the best for dense scenarios and comparable with every other in the more sparse scenarios. Both TS-simple and LS-multi are very close in the results, though. SA as implemented is somewhat inferior, especially for the dense scenarios. It looks as if the dense scenarios had a solutions space more ill-suited for the metaheuristics in general and SA in particular. LS-single was, as expected, the worst and is far from the others in all aspects.

As the results show, none of the methods fulfils the property set up in Section 3 of having the mean value of $C(x)$ to be within 5% of the optimal value for all scenarios. TS-advanced almost makes it, though. In the next section various improvements to help the TSA algorithm to fulfil this criterion will be discussed.

## 8.  Improving the Tabu Search algorithms

The five algorithms described above were all improved in different ways. Best

## 8.1.  No logfile generation

The TS-Advanced version was stripped for every output command, except for the showing of the final assignment plan. When solving the 2-13 scenario a 23.5 % improvement of the mean number of iterations performed was achieved, increasing the sample mean of $C(x)$ by about 0.6% as seen in Table 5. For the 8-13 scenario, the number of iterations performed within the time limit once again was improved by 23.5%, while this meant less to the achieved mean value of $C(x)$, which only raised by 0.2%.

| Scenario | method | $C(x)$ | iterations | *time_opt*'s |
|----------|--------|--------|------------|-------------|
| 2-13 | TSA normal | 235.08 | 497.20 | 8669.40 |
|      | TSA -out | 236.56 | 614.06 | 10725.10 |
| 8-13 | TSA normal | 472.41 | 210.16 | 11805.64 |
|      | TSA -out | 473.31 | 267.86 | 15077.18 |

Table 5. The gain when writing no logfile (-out). The columns show the mean values of the results

## 8.2.  Tabu list clearance

As the Tabu Search algorithms were implemented, the tabu list would never be cleared. When a new best solution were found, a tabu element might therefore block an effective local search performed from this solution, and thus the possibility of finding an even better solution at the next iteration could be missed.

In order to solve this problem the TS-Advanced implementation was changed so that the tabu list was cleared every time a better solution was found. This turned out to make the search for optima more efficient. As it can be seen in Table 6, $C(x)$ was improved by 2.19% (resp. 0.44%) when compared to the original algorithm on the 2-13 (resp. the 8-13) scenario.

| Scenario | method | mean | s. dev. | max | min | % best |
|----------|--------|------|---------|-----|-----|--------|
| 2-13 | TSA | 235.08 | 5.27 | 247.73 | 222.14 | 5.11 |
|      | TSA cl | 240.24 | 5.34 | 247.73 | 224.20 | 3.02 |
|      | TSA cl -out | 240.29 | 5.19 | 247.73 | 227.38 | 3.00 |
| 8-13 | TSA | 472.41 | 5.27 | 480.81 | 457.36 | 2.13 |
|      | TSA cl | 474.47 | 5.73 | 482.62 | 461.33 | 1.70 |
|      | TSA cl -out | 475.89 | 3.73 | 479.81 | 438.11 | 1.40 |

If the output logging were removed too, as earlier described, the results were even better, though not that much.

For both scenarios this means that the wanted quality properties defined in Section 3, which the algorithms should fulfil, now are achieved since the mean values of $C(x)$ are within 5% from the "optimal" value.

## 9.   Quality vs. time usage

With this "final" algorithm (the TS-Advanced with modified *military value* calculation) it was now tested how much the CPU-time used actually meant to the quality of the result. As usual it was tested on both the 2-13 and the 8-13 scenario, but the *military value*-function of the algorithm was changed at this point, to make the results more comparable with those of the actual airforce algorithm. As a result of the change the $C(x)$ values obtained were higher and the standard deviation somewhat lower. Therefore, the results in this section cannot be compared to those previously shown.

The results can be seen in Figs. 12 and 13. The horizontal lines represent the objective value, respectively, 1% and 2% below the best achieved solution.

As shown a 100 times improvement in the computation time is not necessary — a 10 times improvement (i.e. by allowing 30 secs.) still give results far better than the 5% limit initially demanded.
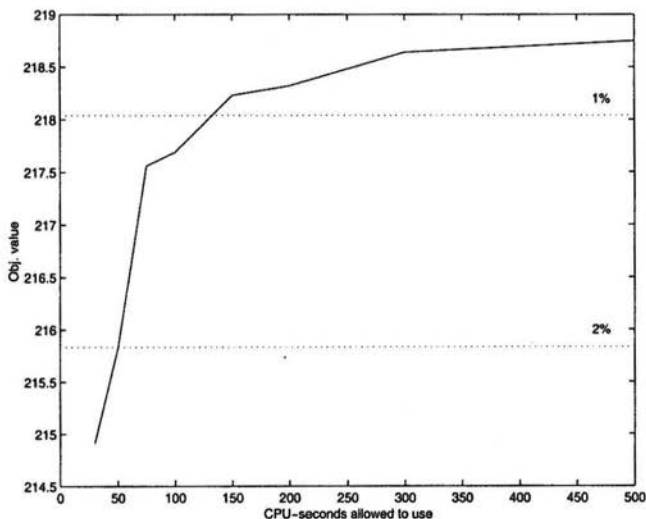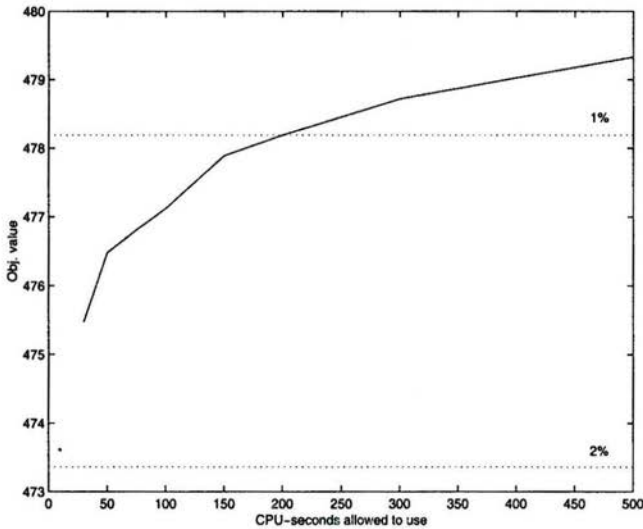
Figure 13. Time vs. quality for TSA used on the 8-13 scenario

## 10.  Conclusions

Due to the complexity of the target-radar assignment problem, the neighbour-
hood generation for both the implemented optimization parts was complicated
and thus time consuming, in view of numerous special cases. In addition, the
time to solve the problem was very limited. So, this was not the most usual
situation to use a metaheuristic approach, though for the same reasons it was
interesting to see the performance of the metaheuristic methods in this uncom-
mon environment.

From the results it is concluded that TS-advanced is the best of the meta-
heuristics implemented with TS-simple and LS-multi close behind. For SA too
little time was available to work properly, probably since cooling should happen
faster than it was effective for the solution space of the problem, especially with
that of high-density scenarios.

The results were compared to those of the officially chosen airforce algo-
rithm. Unfortunately, due to differences in the defined strong and weak criteria
an actual comparison between the TS-advanced algorithm and the airforce al-
gorithm was impossible. The problem was that the defined objective function,
which TS-advanced maximized the best, did not ensure that the targets — if
possible — were intercepted before they were able to fire back. And this spe-
cific property is valued very high by the Danish Airforce. By adapting the time
optimization procedure of the chosen airforce algorithm, this problem could

military value making it punish for crossing the critical line harder than now. None of these changes have been tried, though. More on this topic can be found in Hindsberger (1998-I).

Looking at the criteria defined back in Section 3, the following conclusions can be made:

- Speed: As shown in Section 9 gaining just a 10 times improvement of the computation time will still keep you well within the quality property of 5%. Such a gain should be possible to achieve just by using modern hardware. Implementing in assembler or C++ code would increase speed even more. Assigning the missiles can be done afterwards and should create no troubles as long as you have sufficient missiles for all engagements. If not, a restriction on the number of engagements should be added to the optimization model presented in this paper. The overall speed of the algorithm should not be lowered significantly by adding the missile assignment procedure.

- Maximizing: It was demanded that the expected value of the objective function was within 5% from the global optimum. TS-advanced complied with this property and had, when modified for using the modified military value calculation, a large margin to the 5% demanded for all tested scenarios.

- Range: Whether the critical line was crossed by any targets before being intercepted, was not researched that much. This was because it initially was expected that a maximization of the defined objective function would take care of this, too. As mentioned, it turned out not to be the case, and this is something, which should be worked on in the future.

- Robustness: This property is not relevant when performing a single run, since this is a static situation. None of the metaheuristics implemented were ready to run in a dynamic system updating the assignment plan for every 3-second period. Keeping the solution stable in such a system can be achieved by using the previous solution as the initial for the next run. This solution might need to be repaired at first in order to turn it into a feasible solution. This is an area where further research can be done.

In general, the results were promising and better than expected. So, at least, on this problem — though the problem was not that well suited — a metaheuristic solution could be used with acceptable results. It might therefore also be true for other problems with similar characteristics.

# References

CULLEN, T. and FOSS, C.F., eds. (1997) *Janes Land-based Air Defence 1997-98*. Janes Information Group Ltd., London.

GLOVER, F. and LAGUNA, M. (1993) Tabu Search. In: C.R. Reeves, ed., *Modern Heuristic Techniques for Combinatorial Problems*. Wiley & Sons Inc.

HINDSBERGER, M. (1998-I) The target-radar allocation problem. Master Thesis 98-09, IMM, Technical University of Denmark.

HINDSBERGER, M. (1998-II) Metaheuristics — an introduction. Research Report F-65/1998, Danish Defence Research Establishment, Copenhagen, Denmark.

PIRLOT, M. (1992) General local search heuristics in combinatorial optimization: A tutorial. *Belgian Journal of Operations Research, Statistics, and Computer Science*, **32**, 7-67.

SILVER, E.A., VIDAL, R.V.V. and DE WERRA, D. (1980) A tutorial on heuristic methods. *European Journal of Operations Research*, **5**, 153-162.

VIDAL, R.V.V., ed. (1993) Applied Simulated Annealing. *Lecture Notes in Economics and Mathematical Systems*, **396**. Springer-Verlag.