Autonomous Search and Rescue Rotorcraft Mission Stochastic Planning with Generic DBNs

Florent Teichteil-Königsbuch[1] and Patrick Fabiani[2]

[1] florent.teichteil@cert.fr
[2] patrick.fabiani@cert.fr
ONERA-DCSD
Toulouse, France 31055

1 Introduction

1.1 Motivation

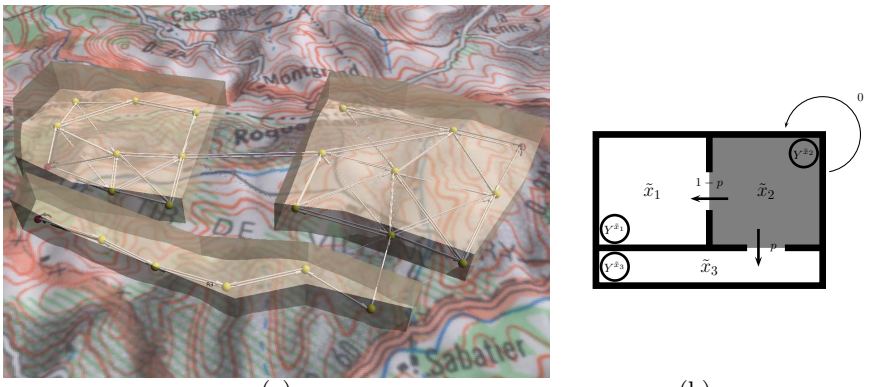
This paper proposes an original generic hierarchical framework in order to facilitate the modeling stage of complex autonomous robotics mission planning problems with action uncertainties. Such stochastic planning problems can be modeled as Markov Decision Processes [5]. This work is motivated by a real application to autonomous search and rescue rotorcraft within the ReSSAC[1] project at ONERA. As shown in Figure 1.a, an autonomous rotorcraft must fly and explore over regions, using waypoints, and in order to find one (roughly localized) person per region (dark small areas). Uncertainties can come from the unpredictability of the environment (wind, visibility) or from a partial knowledge of it: map of obstacles, or elevation map etc. After a short presentation of the framework of structured Markov Decision Processes (MDPs), we present a new original hierarchical MDP model based on generic Dynamic Bayesian Network templates. We illustrate the benefits of our approach on the basis of search and rescue missions of the ReSSAC project.

1.2 Factored Markov Decision Processes

MDPs [5] are a classical model for decision-making under uncertainty. A MDP is a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R} \rangle$ where $\mathcal{S}$ is the set of agent's states, $\mathcal{A}$ is the set of its actions, $\mathcal{P}$ and $\mathcal{R}$ respectively are the markovian probability and reward transitions between states for each action. A solution of a MDP is a mapping $\pi : \mathcal{S} \rightarrow \mathcal{A}$ named *policy*, that can be iteratively computed on the basis of the Bellman's equation [5].

Factored Markov Decision Processes (MDPs) [1, 3] are an extension of MDPs where the state space $\mathcal{S}$ is defined as a cartesian product of n subspaces $\mathcal{V}$ corresponding to an equal number of state variables $\mathcal{S} = \otimes_{i=1}^{n} V_i$. State variable transitions are defined using Dynamic Bayesian Networks (DBNs) [1]. For each

[1] http://www.cert.fr/dcsd/RESSAC/



(a) (b)

Fig. 1. (a) Search and rescue autonomous rotorcraft mission: 3 persons must be rescued in the 3 regions of the navigation subspace (software screenshot). (b) Local policy defined in the region $\tau(\pi) = \tilde{x}_2$. Stochastic outcomes are regions $\zeta(\pi) = \{\tilde{x}_1; \tilde{x}_3\}$.

action, a DBN represents the stochastic dependencies between *post-action* state variables $(X'_i)_{i=1}^n$ and *pre-action* state variables $(X_i)_{i=1}^n$ (see Figure 6.a).

For each post-action state variable X_i, a probability tree encodes the stochastic distribution of X'_i values (tree's leafs) knowing the other state variables values (nodes), as shown in Figure 6.b. The reward transitions are encoded as a single decision tree for each action. Classical MDP optimization algorithms are generalized in structured algorithms [1, 3].

1.3 A hierarchical approach

Modeling autonomous robotics problems with factored MDPs remains difficult. In the very simple search and rescue mission of Figure 1.a, with 5 actions: `west`, `east`, `north`, `south`, `statio`, and 4 state variables: the rotorcraft's localization and the status of the 3 persons to rescue, the localization variable has 24 possible values (as many as the number of waypoints), that must be enumerated in any decision tree containing a waypoint node. More complicated missions can have hundreds of waypoints, which makes it a burden to model *by hand* the problem because the trees' sizes are polynomial in the arity of state variables.

Our hierarchical model allows to tackle larger state spaces by reducing the size of the decision trees used to model the problem. We use state abstractions in order to decompose the problem with respect to its variables of highest arity: in the search and rescue example of Figure 1, the localization variable (24 positions) is decomposed into a region variable of arity 3.

2 Hierarchical factored MDP

2.1 State subspace splitting

Let X_p be a state variable with a large arity. The state subspace generated by X_p (navigation subspace) is a graph V_p that can be partitioned into smaller weakly coupled abstract subgraphs $\tilde{V}_p$. The partition can be either a mission input, or the result of an automatic partition process [6]. The resulting abstracted states can be considered as the values of a new abstracted state variable $\tilde{X}_p$, which is an abstraction of the original state variable X_p. The abstract state space of the factored MDP becomes $\tilde{\mathcal{V}} = (\otimes_{i \neq p} V_i) \times \tilde{V}_p$. Let us consider the mission of Figure 1: whereas a X_p node would have 24 subtrees, the corresponding $\tilde{X}_p$ node only has 3 subtrees.

2.2 Local policies

Actions need to be abstracted correspondingly into macro-actions. At the region level, abstract actions correspond to local policies defined and applied within the regions of the partition $\tilde{V}_p$. Let π be such a local policy, defined in a region $\tilde{v}_p$. Let Π_p be a set of local policies defined on each region of the partition $\tilde{V}_p$. A minimal set of local policies can be automatically generated [2, 4], in such a way that an optimal policies can be obtained as a combination of such local policies in the regions. Extra local policies can be added by other methods.

Unfortunately, in both cases, the number of local policies can be very large. In theory, the maximum number of local policies is $\sum_{\tilde{v}_p \in \tilde{V}_p} |\mathcal{A}|^{|\tilde{v}_p|}$, each of which should have a corresponding DBN encoding for the dependencies between the pre- and post-action variables.

In order to keep a substantial benefit from the decomposition, it is useful to notice that in most problems, all the local policies DBNs share a common structure. It is indeed possible to define a single DBN structure, where the corresponding local policy, the region where it is applicable, and the reachable regions appear as parameters that can be automatically instantiated when the local policies are computed.

3 Abstract generic Dynamic Bayesian Network

In this section, we present the syntax of our abstract generic DBN for modeling factored stochastic autonomous robotics problems. Our generic DBN is parametrized by a local policy $\pi \in \Pi_p$. Since a local policy is defined for a single region of the reduced variable $\tilde{X}_p$, we can define the mapping $\tau : \Pi_p \rightarrow \tilde{V}_p$ between local policies and the region where they are each one defined.

We illustrate our approach with a small academic instance of a search and rescue autonomous rotorcraft mission (see Figure 1.b). The decomposed subspace matches the localization variable, whose arity is 24, abstracted in 3 regions ($\tilde{X}$). We will consider a local policy π defined in the second region $\tilde{x}_2$,

that consists in going out towards the regions $\tilde{x}_1$ and $\tilde{x}_3$ with respectively the probabilities $1-p$ and p. Last but not least, each region contains a person to rescue: these subgoals are represented by 3 binary state variables $\left(Y^{\tilde{x}_i}\right)_{1 \leqslant i \leqslant 3}$ indicating if each person was already rescued or not.

3.1 Reduced state variable modeling

Let us consider a decision tree (probability tree or reward tree) containing a node of the reduced variable $\tilde{X}_p$. The local policy π is only defined in $\tau(\pi)$ so that the node $\tilde{X}_p$ only has two abstract subtrees: one corresponding to the value $\tau(\pi)$, and one other representing the other values where the policy is not applicable, noted $\overline{\tau(\pi)} = \tilde{V}_p \setminus \{\tau(\pi)\}$.

Since π is only applicable over $\tau(\pi)$, the $\overline{\tau(\pi)}$-subtree of any $\tilde{X}_p$ variable in probability trees is symbolically represented as a nil leaf. Instead of defining these nil leafs inside each probability tree, it is better to define a binary mask tree that indicates where the local policy is applicable. This mask tree should contain at least a node of the state variable $\tilde{X}_p$, as shown in Figure 2.

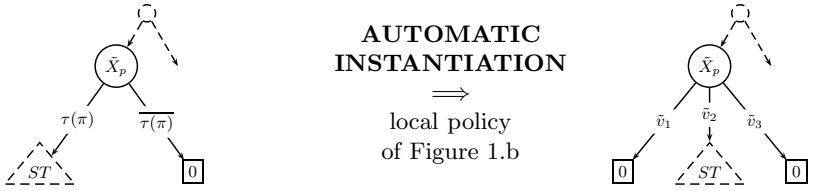


Fig. 2. Generic mask tree example and one of its instantiations

The function that automatically instantiates the subtrees of a $\tilde{X}_p$ node in any decision tree is presented in Algorithm 1. It calls the function `InstantiateTree`, that instantiates the $\tau(\pi)$-subtree of the generic node $\mathcal{T}$ (see Algorithm 6). The $\overline{\tau(\pi)}$-subtrees are nil leafs (nil_leaf).

Algorithm 1: Function `InstantiateXpSubtrees`

Data: $\mathcal{T}$ (generic node), $\mathcal{T}^\pi$ (instantiated node), π, τ, ζ, $[\tilde{v}'_p = -1]$
Result: $\mathcal{T}^\pi$ (instantiated tree)
begin
  $subtree \leftarrow \mathcal{T}.son(\text{`}\tau(\pi)\text{'})$;
  for $\tilde{v}_p \in \tilde{V}_p$ **do**
    if $\tilde{v}_p = \tau(\pi)$ **then** $\mathcal{T}^\pi.sons().push(\texttt{InstantiateTree}(subtree, \pi, \tau, \zeta, \tilde{v}'_p))$;
    else $\mathcal{T}^\pi.sons().push(nil\_leaf)$;
  return $\mathcal{T}^\pi$;
end

The treatment of a $\tilde{X}'_p$ node is slightly different from a $\tilde{X}_p$ node. Let $\zeta : \Pi_p \rightarrow \tilde{V}_p$ be the mapping from a local policy to its reachable regions. It means that π transforms $\tau(\pi)$ into $\zeta(\pi)$. In our small instance depicted in Figure 1.b, only the regions $\tilde{x}_1$ and $\tilde{x}_3$ are reachable with $\pi : \zeta(\pi) = \{\tilde{x}_1; \tilde{x}_3\}$.

A $\tilde{X}'_p$ node can only have 2 abstract subtrees: one for the value $\zeta(\pi)$ and one other for the value $\overline{\zeta(\pi)} = \tilde{V}_p \setminus \zeta(\pi)$. Each subtree must be transformed into as many subtrees as the cardinality of the corresponding abstract value (see Figure 3 and Algorithm 2).

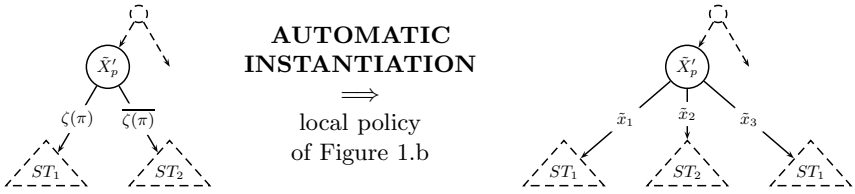


Fig. 3. Example of a decision tree containing a $\tilde{X}'_p$ node and one of its instantiations

Algorithm 2: Function `InstantiateXppSubtrees`

Data: $\mathcal{T}$ (generic node), $\mathcal{T}^\pi$ (instantiated node), π, τ, ζ
Result: $\mathcal{T}^\pi$ (instantiated tree)
begin
  $st_1 \leftarrow \mathcal{T}.son(\text{`}\zeta(\pi)\text{'})$;
  $st_2 \leftarrow \mathcal{T}.son(\text{`}\overline{\zeta(\pi)}\text{'})$;
  for $\tilde{v}'_p \in \tilde{V}_p$ **do**
    if $\tilde{v}'_p \in \zeta(\pi)$ **then** $\mathcal{T}^\pi.sons().push(\texttt{InstantiateTree}(st_1, \pi, \tau, \zeta, \tilde{v}'_p))$;
    else $\mathcal{T}^\pi.sons().push(\texttt{InstantiateTree}(st_2, \pi, \tau, \zeta, \tilde{v}'_p))$;
  return $\mathcal{T}^\pi$;
end

3.2 State variables depending on the reduced state variable

We can take advantage of our abstract model to introduce state variables that are defined for each value of the reduced state variable. In the case of our small exploration mission (Figure 1), let us consider a person to rescue in each region of the navigation subspace. Each value $\tilde{v}_p$ of the abstract navigation state variable corresponds to a subgoal to achieve, represented by a binary state variable $Y^{\tilde{v}_p}$ (see Figure 1.b).

Only can be achieved the subgoal corresponding to the region where the unknown local policy of our generic DBN is defined. The other subgoals can not be realized with this local policy, since it is not applicable inside the regions where

they are enclosed. Therefore, for each set $\left(Y^{\tilde{v}_p}\right)_{\tilde{v}_p \in \tilde{V}_p}$ of variables depending on the reduced variable, the generic DBN defines 2 abstract variables: the variable $Y^{\tau(\pi)}$ defined for the abstract value $\tau(\pi)$, and the variable $Y^{\overline{\tau(\pi)}}$ representing all the variables defined in the regions $\overline{\tau(\pi)}$.

Figure 4 depicts the decision trees of $Y^{\tau(\pi)}$ and $Y^{\overline{\tau(\pi)}}$ and an instance of their automatic instantiation for a given local policy. A decision tree containing a $Y^{\tau(\pi)}$ node is illustrated too.

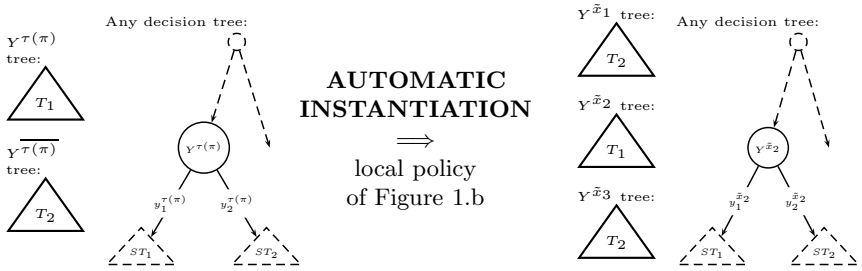


Fig. 4. Example of the decision trees of $Y^{\tau(\pi)}$ and $Y^{\overline{\tau(\pi)}}$, and of a decision tree containing a $Y^{\tau(\pi)}$ node. An automatic instantiation is presented.

Algorithm 3 details the automatic instantiation of the two abstract probability trees $\mathcal{T}_{Y^{\tau(\pi)}}$ and $\mathcal{T}_{Y^{\overline{\tau(\pi)}}}$. Since a node of any decision tree can be an abstract $Y^{\tau(\pi)}$ node (primed or not), it must be analyzed before being instantiated, as done in Algorithm 4.

Algorithm 3: Function `InstantiateYpTrees`

Data: $\mathcal{T}_{Y^{`\tau(\pi)'}}$, $\mathcal{T}_{Y^{`\overline{\tau(\pi)}'}}$, π, τ, ζ
Result: $\left(\mathcal{T}^{\pi}_{Y^{\tilde{v}_p}}\right)_{\tilde{v}_p \in \tilde{V}_p}$
$\mathcal{T}^{\pi}_{Y^{\tau(\pi)}} \leftarrow$ `InstantiateTree`$(\mathcal{T}_{Y^{`\tau(\pi)'}}, \pi, \tau, \zeta)$;
for $\tilde{v}_p \in \overline{\tau(\pi)}$ **do** $\mathcal{T}^{\pi}_{Y^{\tilde{v}_p}} \leftarrow$ `InstantiateTree`$(\mathcal{T}_{Y^{`\overline{\tau(\pi)}'}}, \pi, \tau, \zeta)$;

3.3 Abstract leafs of the generic probability trees

Due to action uncertainties, the outcome of a local policy is not deterministic. Let us consider for instance the local policy depicted in Figure 1.b: starting from region $\tilde{x}_2$, the local policy can lead to regions $\tilde{x}_1$ and $\tilde{x}_3$ with respectively probabilities $1-p$ and p. Let $\tilde{\mathcal{P}}^{\pi}$ be the abstract probability transition distribution over the partitioned subspace $\tilde{V}_p$ for the local policy π: this distribution is the stationary probability distribution of the markov chain resulting from application of the local policy π inside $\tau(\pi)$ [2].

Algorithm 4: Function `InstantiateNode`

Data: n (generic node), π, τ
Result: n^{π} (instantiated node)
begin
  if $n = Y^{`\tau(\pi)'[']}$ **then** $n^{\pi} \leftarrow Y^{\tau(\pi)[']}$;
  else $n^{\pi} \leftarrow n$;
  return n^{π};
end

The probabilities of obtaining the different values of any state variable may depend on the local policy probability distribution. These state variable probabilities are stored in the leafs of their probability trees. We suppose that they can be expressed as functions of 2 abstract local policy probabilities:

- $p^{\tau(\pi)}$: probability of staying in the region $\tau(\pi)$
- $p^{\zeta(\pi)}$: if the reduced post-action state variable ($\tilde{X}'_p$) is a parent node, probability of going to the value of the parent reduced state variable

An example of abstract probability leaf and one of its possible instantiations are shown in Figure 5. The abstract leaf is a formal algebraic expression of $p^{\tau(\pi)}$ and $p^{\zeta(\pi)}$. Given the abstract probability transition distribution $\tilde{\mathcal{P}}^{\pi}$ over the partitioned subspace $\tilde{V}_p$ for the local policy π, Algorithm 5 computes the probability of an instantiated leaf. It calls the function `Evaluate` from the computer algebra library to assess the leaf. If $\tilde{v}'_p \neq -1$, it means that $\tilde{X}'_p$ is a parent node of the leaf l, and l belongs to the $\tilde{v}'_p$-subtree of the $\tilde{X}'_p$ parent node.

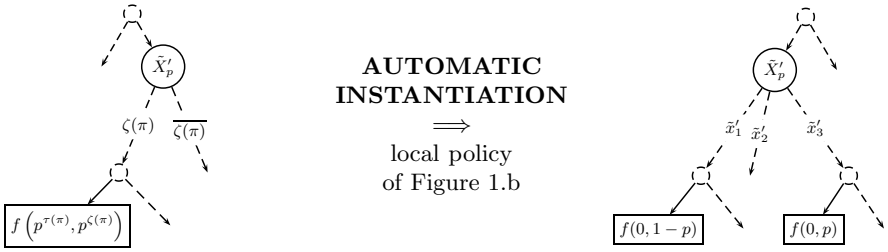


Fig. 5. Generic probability leaf example and one of its instantiations

3.4 Abstract leafs of the generic reward tree

Local policy transition probabilities are associated with local policy transition rewards. Let $\tilde{\mathcal{R}}^{\pi}$ be the transition rewards defined for the local policy π over the reduced state variable subspace. These reward transitions can be computed on the basis of the local policy transition probabilities just defined.

Algorithm 5: Function `InstantiateLeaf`

Data: l (generic leaf), π, τ, ζ, $[\tilde{v}'_p = -1]$
Result: l^π (instantiated leaf)
$p^{\tau(\pi)} \leftarrow \tilde{\mathcal{P}}^\pi(\tau(\pi), \tau(\pi))$;
if $\tilde{v}'_p \neq -1$ **then** $p^{\zeta(\pi)} \leftarrow \tilde{\mathcal{P}}^\pi(\tau(\pi), \tilde{v}'_p)$;
$l^\pi \leftarrow$ `Evaluate`$(l, `p^{\tau(\pi)}` = p^{\tau(\pi)}, [`p^{\zeta(\pi)}` = p^{\zeta(\pi)}])$;

As for the local policy transition probabilities, we suppose that the local policy transition rewards are formal algebraic expressions of:

- $r^{\tau(\pi)}$: average reward obtained if staying in $\tau(\pi)$
- $r^{\zeta(\pi)}$: if the reduced post-action state variable ($\tilde{X}'_p$) is a parent node, average reward if going to the value of the parent reduced state variable

Figure 5 still is a good example of a generic reward tree and its instantiation for the local policy of Figure 1.b, with the proviso of replacing $p^{\cdot}$ by $r^{\cdot}$. In the same way, Algorithm 5 presents the automatic reward leaf instantiation algorithm, with the proviso of replacing $p^{\cdot}$ by $r^{\cdot}$ and $\tilde{\mathcal{P}}$ by $\tilde{\mathcal{R}}$.

3.5 Main automatic DBN instantiation algorithm

Algorithm 6 automatically instantiates a decision tree for a given local policy. The version of our algorithm presented in this paper is recursive. It is called from functions `InstantiateXpSubtrees` and `InstantiateXppSubtrees`, when instantiating the subtrees of the nodes $\tilde{X}_p$ and $\tilde{X}'_p$ (see Algorithms 1 and 2). Notice that the optional argument $\tilde{v}'_p$ is not an input of `InstantiateXppSubtrees`: otherwise, it would mean that $\tilde{X}'_p$ is a parent node of itself, what is impossible.

4 Application to a search and rescue mission

We applied our generic MDP model to search and rescue missions described in section 1.1. We tested our generic model with 4 state variables (see Figure 6):

- R : regions of the environment (stands for $\tilde{X}_p$)
- $O.$: person to rescue in the region where the unknown local policy is defined (stands for $Y^{`\tau(\pi)'}$)
- $O_{\_}$: persons to rescue in the other regions (stands for $Y^{`\overline{\tau(\pi)}'}$)
- A : rotorcraft's autonomy (binary variable, `full` or `empty`)

In '$O.$' probability tree leafs, $Lp.$ stands for '$p^{\tau(\pi)}$' and $Lp_{\_} = 1 - Lp. = p^{\overline{\tau(\pi)}}$.

Table 1 shows the elapsed time comparison between automatic instantiation and optimization stages, when increasing the sizes of both the state and action spaces. Note that **the same generic DBN** was used to model all of the tested

Algorithm 6: Function `InstantiateTree` (recursive)

Data: $\mathcal{T}$ (generic decision tree), π, τ, ζ, $[\tilde{v}'_p = -1]$
Result: $\mathcal{T}^\pi$ (instantiated decision tree)
begin
  if $\mathcal{T}.root().type() = leaf$ **then** $\mathcal{T}^\pi \leftarrow$ `InstantiateLeaf`$(\mathcal{T}.root(), \pi, \tau, \zeta, \tilde{v}'_p)$;
  else
    $\mathcal{T}^\pi \leftarrow$ `InstantiateNode`$(\mathcal{T}.root(), \pi, \tau)$;
    switch $\mathcal{T}.root()$ **do**
      case $\tilde{X}_p$: $\mathcal{T}^\pi \leftarrow$ `InstantiateXpSubtrees`$(\mathcal{T}.root(), \mathcal{T}^\pi, \pi, \tau, \zeta, \tilde{v}'_p)$;
      case $\tilde{X}'_p$: $\mathcal{T}^\pi \leftarrow$ `InstantiateXppSubtrees`$(\mathcal{T}.root(), \mathcal{T}^\pi, \pi, \tau, \zeta)$;
      otherwise
        for $subtree \in \mathcal{T}.root().sons()$ **do**
          $\mathcal{T}^\pi.sons().push($`InstantiateTree`$(subtree, \pi, \tau, \zeta, \tilde{v}'_p))$;
  return $\mathcal{T}^\pi$;
end

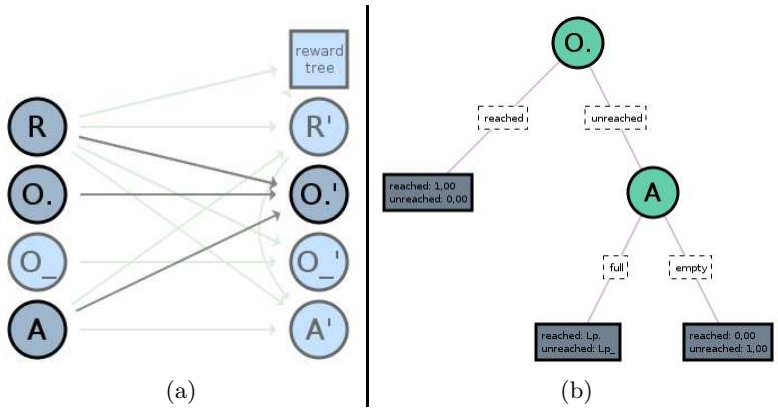


Fig. 6. (a) Generic DBN and (b) $O.$ generic probability tree (software screenshot)

Nb of enumerated states	Nb of regions (states per region)	Nb of generated local policies	**DBNs instantiation time**	MDP optimization time
82944	9 (9)	21	**0.01**	0.12
746496	9 (81)	61	**0.01**	16.77
58982400	17 (9)	69	**0.03**	1621.62
530841600	17 (81)	117	**0.06**	$>$ 1 hour

Table 1. Elapsed time comparison between instantiation and optimization stages, for growing size search and rescue missions (in seconds, with a P4-2.8GHz processor)

instances. First, it appears that the number of states exponentially grows with number of regions, so that unstructured enumerated models of MDP would have been very tedious and quite impossible to model. Second, the number of generated local policies (automatic generation algorithm of [4]) is round 100, what means that usual factored MDP models would have required to manually input a hundred or even more DBNs, in order to define our real search and rescue missions. On the contrary, our generic hierarchical DBN model enables to define **only one DBN** for the whole mission. Third, the automatic DBNs intanciation time is insignificant compared to the optimization time ($< 1\%$): it confirms the modeling and effectiveness benefits of our approach.

5 Conclusion

In this paper, we proposed an original generic hierarchical framework for modeling large factored Markov Decision Processes. Our approach is based on a decomposition into regions of the state subspaces engendered by the state variables with large arity. The regions are macro-states of the thus abstracted MDP. Local policies can then be computed (or defined by other means) in each region of the decomposition and taken as macro-action of the abstract MDP. The factored MDP model is then defined at the abstract level. A generic DBN template can be defined, symbolically parametrized by the local policies. We illustrated and showed the significance of our method on real instances of search and rescue aerial robotics missions (within the ReSSAC project) where the navigation subspace can easily be decomposed into regions: the use of classical unstructured MDP models would have been very tedious and perhaps impossible for the kind of real planning missions we tackle.

References

1. Craig Boutilier, Richard Dearden, and Moises Goldszmidt. Stochastic dynamic programming with factored representations. *Artificial Intelligence*, 121(1-2):49–107, 2000.
2. Milos Hauskrecht, Nicolas Meuleau, Leslie Pack Kaelbling, Thomas L. Dean, and Craig Boutilier. Hierarchical solution of markov decision processes using macro-actions. In *Proceedings 14th UAI*, pages 220–229, San Francisco, CA, 1998.
3. Jesse Hoey, Robert St-Aubin, Alan Hu, and Craig Boutilier. Optimal and approximate stochastic planning using decision diagrams. Technical Report TR-2000-05, University of British Columbia, 10 2000.
4. Ron Parr. Flexible decomposition algorithms for weakly coupled markov decision problems. In *Proceedings 14th UAI*, pages 422–430, San Francisco, CA, 1998.
5. Martin L. Puterman. *Markov Decision Processes.* John Wiley & Sons, INC, 1994.
6. R. Sabbadin. Graph partitioning techniques for markov decision processes decomposition. In *Proceedings 15th ECAI*, pages 670–674, Lyon, France, July 2002.

Solving multi-objective scheduling problems—An integrated systems approach

Martin Josef Geiger

Lehrstuhl für Industriebetriebslehre, Universität Hohenheim, D-70593 Stuttgart, Germany, mjgeiger@uni-hohenheim.de

Abstract. In the past, numerous approaches have been formulated either for approximating Pareto-optimal alternatives or supporting the decision making process with an interactive multi criteria decision aiding methodology. The article on the other hand presents an integrated system for the resolution of problems under multiple objectives, combining both aspects. A method base of metaheuristics is made available for the identification of optimal alternatives of machine scheduling problems, and the selection of a most preferred solution is supported in an interactive decision making procedure.
As the system is aimed at end users, a graphical interface allows the easy adaptation of metaheuristic techniques. Contrary to existing software class libraries, the system therefore enables users with little or no knowledge in the mentioned areas to successfully solve scheduling problems and customize and test metaheuristics.
After successfully competing in the finals in Ronneby (Sweden), the software has been awarded the *European Academic Software Award 2002* (http://www.easa-award.net/, http://www.bth.se/llab/easa_2002.nsf).

Key words: Multi-Objective Optimization, Multi-Objective Metaheuristics, Decision Support System, Scheduling

1 Introduction

In general, the resolution of multi-objective problems is twofold. First, optimal solutions need to be identified by means of some algorithmic approach. For a given problem Π having a set of feasible alternatives $\mathcal{X}$ and a set of optimality criteria $G(x) = (g_1(x), \ldots, g_k(x)), x \in \mathcal{X}$, optimality of alternatives is in the light of conflicting optimality criteria here understood in the sense of *Pareto-optimality* [20] as further described in Definition 1 and 2. Without loss of generality, a minimization of the objective function values is assumed here.

Definition 1 (Dominance). *An objective vector $G(x), x \in \mathcal{X}$ is said to dominate a vector $G(x'), x' \in \mathcal{X}$ if and only if $g_i(x) \leq g_i(x')\ \forall\ i = 1, \ldots, k \wedge \exists i \mid g_i(x) < g_i(x')$. The dominance of $G(x)$ over $G(x')$ is denoted with $G(x) \prec G(x')$.*

Definition 2 (Efficiency, Pareto-optimality, Pareto set). *An objective vector* $G(x), x \in \mathcal{X}$ *is said to be* efficient*, if and only if* $\neg\exists x' \in \mathcal{X} \mid G(x') \prec G(x)$*. The corresponding alternative* x *is called* Pareto-optimal*, the set of all Pareto-optimal alternatives the* Pareto set P*.*

The second step of the resolution of multi-objective problems is the selection of a most preferred alternative $x^* \in P$, involving a single human decision maker or even a group of people.

Search and *decision making* can be combined in three general ways [6].

1. A priori: The decision maker states his/her preferences allowing the construction of a utility function and the successive resolution of the resulting mono-criterion problem of maximizing the overall utility.
2. Interactive: The resolution of the problem alternates between search and decision making, successively revealing the preferences of the decision maker.
3. A posteriori: The choice of a most preferred alternative is performed after the determination of all optimal solutions.

Over the years, numerous concepts have been proposed to support both aspects of search and decision making. Most interactive methods are based on goal programming [13] or reference point approaches [23] and allow the successive refinement of the decision makers' preferences. An overview is e. g. given by Vincke [21].

Besides methodological progress, implementations of algorithms have been made freely available on the world wide web. For genetic algorithms for example, an archive is maintained under `http://www.aic.nrl.navy.mil/galist/src/`. The particular case of multi-objective optimization has been addressed by several researchers, and an overview about implemented source code is maintained by Coello Coello on the EMOO webpage, `http://www.lania.mx/~ccoello/EMOO/EMOOsoftware.html`. Unfortunately however, most implementations require a throughout understanding of the underlying methodologies and techniques in order to be reused and adapted to particular problem domains. This can impose a problem in teaching and demonstration work, when non-experts are required to interact with the computer programs. Here, implementations are required similar to established computer user interfaces with which the users are familiar. Only very recently, components are being developed that allow the visualization of the outcomes of multi-objective optimization problems, one example being GUIMOO by Cahon, Van Den Hekke and Seynhaeve. Integrated systems however, combining both search *and* decision making are to our knowledge not freely available yet.

The paper is organized as follows. Section 2 presents an integrated system for multi-objective optimization and decision making, using the example of scheduling under multiple objectives. The problem is well-known from operations research and computer science and is of high practical value with applications in many areas [15]. Results obtained with the system are presented in Section 3, and conclusions and discussion are given in Section 4.

2 A metaheuristic system for multi-objective scheduling

2.1 The addressed problem

Machine scheduling considers in general the assignment of a set of resources (machines) $\mathcal{M} = \{M_1, \ldots, M_m\}$ to a set of jobs $\mathcal{J} = \{J_1, \ldots, J_n\}$, each of which consists of a set of operations $J_j = \{O_{j1}, \ldots, O_{jo_j}\}$ [4]. The operations O_{jk} typically may be processed on a single machine $M_i \in \mathcal{M}$ involving a non-negative processing time t_{jk}. Usually, precedence constraints are defined among the operations of a job, reflecting its technical nature of processing. Other important aspects that frequently have to be taken into consideration are release dates and due dates of jobs.

A solution $x \in \mathcal{X}$ to the problem, a so called *schedule*, assigns start and end times for the operations with respect to the defined constraints of the problem.

While first approaches to machine scheduling consider optimality of schedules for a single objective function, multi-objective formulations of the problem have become increasingly of importance in the last years [18]. As these criteria are often conflicting, not a single but a whole set of solutions may be regarded as optimal in the sense of Pareto-optimality, introduced earlier in Definition 2, and the resolution of the problem lies in the identification of all $x \in P$.

Various optimality criteria are based on the completion times C_j of the jobs J_j in the schedule. The most prominent to mention is the minimization of the maximum completion time (makespan) $C_{max} = \max\{C_1, \ldots, C_n\}$. Another objective is the minimization of the sum of the completion times $C_{sum} = \sum_{j=1}^{n} C_j$. Both measures implicitly try to optimize cost of production by minimizing the production time of the jobs.

In many situations, due dates d_j are present for each job J_j which define a preferable or required time of job completion. It is here possible to compute due date violations in the form of tardiness values $T_j = \max\{C_j - d_j, 0\}$. Usual optimality criteria based on this consideration are the minimization of the maximum tardiness $T_{max} = \max\{T_1, \ldots, T_j\}$, the minimization of the total tardiness $T_{sum} = \sum_{j=1}^{n} T_j$, and the minimization of the number of tardy jobs $U = \sum_{j=1}^{n} U_j$ where $U_j = 1$ if $T_j > 0$, 0 otherwise.

In terms of machine efficiency, idle times I_i of the machines M_i may be considered up to the completion of the last job on M_i. Possible optimality criteria are therefore the minimization of the maximum machine idleness $I_{max} = \max\{I_1, \ldots, I_m\}$, and the minimization of the total machine idleness $I_{sum} = \sum_{i=1}^{m} I_i$.

An important factor for the resolution of the scheduling problem is the *regularity* of the functions [5]. It is here possible to represent an optimal schedule as a permutation of operations, corresponding to the position of the job in the sequence of production. An interpretation of the permutations is possible by computing a schedule with respect to the given job sequence, assuming earliest possible execution of the operations [7].

2.2 Components

For the resolution of multi-objective production scheduling problems, the integrated system *MOOPPS* has been implemented. As illustrated in Figure 1, the system consists of different components for the resolution of the problem.

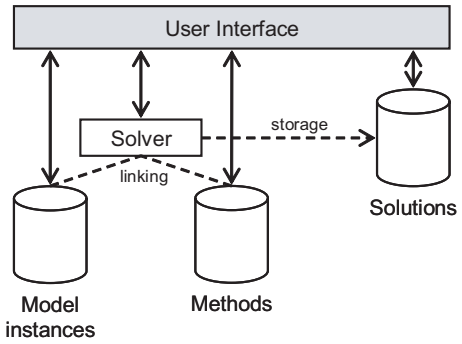


Fig. 1. System architecture.

A *method database* contains a set of heuristics approaches for solving multi-objective scheduling problems:

1. Priority rules [12], based on the early work of Giffler and Thompson [11] for generating active schedules.
2. Local search neighborhoods [16] within a multi-point hillclimber.
3. Multi-objective evolutionary algorithms [1], incorporating elitist strategies and a variety of crossover neighborhoods like e. g. uniform order based crossover, order based crossover, two point order crossover, and partially mapped crossover.
4. The 'MOSA' multi-objective simulated annealing algorithm of Teghem et al. [19].
5. A module based on the 'AIM' aspiration interactive method [14] for an interactive search in the obtained results.

The *model instance database* stores the data of the problem instances that have to be solved. General job shop as well as flow shop scheduling problems can be formulated. Besides newly generated data sets, well-known test instances from literature [3] have been included. Solutions are obtained by linking model instances with methods and stored in a solution database. This allows the reuse of specific metaheuristics for a range of problem instances as well as the comparison of results obtained from different heuristic approaches.

A graphical *user interface* links the modules described above into a single system. Besides the construction of model instances and the definition of metaheuristic search algorithms, an interactive decision making procedure enables the user to select a most preferred schedule. Visualization of alternatives is

available in alternative space and in outcome space. The alternatives as such are presented as Gantt charts [10], their outcomes as plotted Pareto fronts.

2.3 Visual interface

The user interface is a key aspect of the implemented system as it is designed for end users. It brings together visual components to store, retrieve and modify problem instances, configure metaheuristic methods, execute them and manage the obtained results. All components like for example crossover/mutation operators and their corresponding application probabilities are available. New configurations of metaheuristics can be derived from predefined and implemented techniques by simply changing the attribute values of the methods. Also, the progress of the metaheuristics while optimizing a particular problem instance can be monitored by storing the currently best alternatives after definable intervals of evaluations.

After executing methods on problem instances, solutions are obtained that need to be further investigated. As mentioned above, two visualizations are of importance. First, a two-dimensional outcome space plot visualizing the Pareto front. Here, a direct interaction is possible by allowing the user to select alternatives. Second, a visualization of the alternatives as such using job-oriented or machine-oriented Gantt charts [24]. The detailed starting times of the operations can be monitored here, and an indication whether a job is tardy or not is easily available.

In order to allow a widespread use of the software, the graphical user interface is available in English, French, German, Hungarian, Italian, Polish, and Spanish language. Also, a 103-pages printed documentation is available.

2.4 Optimization and decision making

The resolution of multi-objective scheduling problems is supported by a two-stage a posteriori procedure as described in Section 1. First, Pareto-optimal alternatives or an approximation P_a of the Pareto set P are computed using the chosen metaheuristics. Second, an interactive search in the obtained results is performed by the decision maker.

During this interactive decision making procedure, aspiration levels $A = \{a_{g_1}, \ldots, a_{g_k}\}$ for each of the optimized objective functions $G(x) = (g_1(x), \ldots, g_k(x))$ are chosen. As shown in Figure 2, the elements of the approximation P_a of the Pareto set P are accordingly divided into two subsets, the subset P_{as} of the alternatives fulfilling the aspiration levels $(g_i(x) \leq a_{g_i} \forall i = 1, \ldots, k)$ and the subset $P_{\neg as}$ of the alternatives that do not meet the aspiration levels. It is obvious that $P_{as} \cup P_{\neg as} = P_a$ and $P_{as} \cap P_{\neg as} = \emptyset$.

The initial values of the aspiration levels a_{g_i} are set to the worst values in P_a: $a_{g_i} = \max_{x \in P_a} (g_i(x)) \, \forall i = 1, \ldots, k$ and as a consequence, $P_{as} = P_a$. The decision maker is allowed to modify the values of the aspiration levels and successively reduce the number of elements in P_{as} until $|P_{as}| = 1$. The

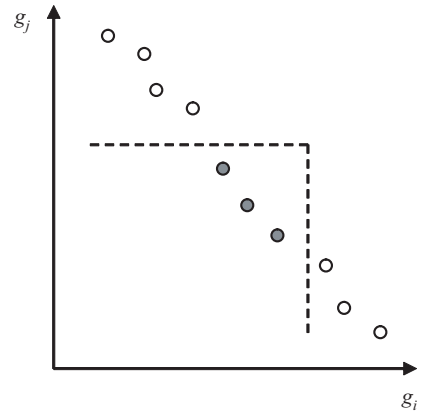


Fig. 2. Decision making component. A cone in outcome space divides the set of alternatives into alternatives fulfilling the aspiration levels (grey background) and alternatives outside the cone (white background).

remaining alternative in P_{as} is the desired compromising solution x^* as the fixed aspiration levels are met by this alternative.

Another interpretation of the method can be seen in a moving cone in outcome space that contains all alternatives fulfilling the current aspiration levels. While the visualization of the outcome space provided by the system is limited to two dimensions at a time, the decision making procedure as such can accommodate higher dimensions without any further problems. The procedure here allows the arbitrary change of the aspiration levels a_{g_i} in any direction, enlarging or reducing them. This is important as the situation in which $P_{as} = \emptyset$ appears, an adjustment of at least one aspiration level a_{g_i} is necessary in order to allow the identification of a most preferred alternative x^*.

3 Computational results

Different configurations of the implemented metaheuristics have been tested on benchmark instances of multi-objective machine scheduling problems taken from literature [1, 2], ranging from $n = 20$ jobs on $m = 5$ machines up to $n = 100$ jobs on $m = 20$ machines. Using various configurations of evolutionary algorithms, close approximations of the true Pareto fronts or the best known solutions could have been obtained.

Algorithm 1 gives the pseudo-code of the applied evolutionary method. It can be seen, that two sets are maintained during search. First, a population POP_i of n_{pop} individuals, and second, an archive P_a of alternatives which are currently not dominated by any other known alternative. After termination of the optimization runs, P_a is returned as an approximation of the true Pareto set

P. This strategy allows the evolvement of the population POP_i while keeping the best found alternatives in P_a, implicitly implementing an elitist strategy.

Algorithm 1 Multi-objective evolutionary algorithm

1: Set $i = 1$
2: Set $P_a = \emptyset$
3: Initialize: Compute starting population POP_i with n_{pop} individuals
4: Update P_a with POP_i
5: **repeat**
6: **repeat**
7: Select alternatives $x_1, x_2 \in POP_i \cup P_a$
8: Compute x'_1, x'_2 using crossover neighborhood $\mathbf{N}_c(x_1, x_2, z)$
9: Apply mutation $\mathbf{N}_m(x, z)$ with probability p_{mut}
10: Test new alternatives x'_1, x'_2 for acceptance in POP_{i+1}
11: **until** number of elements in $P_{i+1} = n_{pop}$
12: Update P_a with P_{i+1}
13: Set $i = i + 1$
14: **until** termination criterion has been met
15: Return P_a

When selecting two alternatives x_1, x_2 for reproduction, the union of both sets POP_i and P_a forms a mating pool. Selection is done with respect to the Pareto-ranking-based approach of Fonseca and Fleming [8]. Crossover operators tested include partially mapped crossover PMX, order based crossover OBX, uniform order based crossover UOBX, and two-point crossover TPOX [22]. New alternatives are generated until a new population POP_{i+1} has been formed which replaces the old population POP_i. Step 10 of Algorithm 1 ensures that no duplicates are added to the succeeding population POP_{i+1}.

The length of the test runs has to be chosen depending on the size of the problem instances. Good termination criteria turned out to be 1,000,000 evaluated alternatives for instances with $n = 20$, 5,000,000 evaluations for instances with $n = 50$, and 10,000,000 for $n = 100$.

For the instances proposed by Basseur et al. [2] on the basis of Taillard [17], the approximations came close to the best known alternatives of which most have been identified. Unfortunately it was not possible to improve any of them. It may be mentioned however, that for the smaller instances the known results are already proven to be optimal and therefor not further improvable.

New alternatives have been identified dominating the previously reported best known solutions for the instance of Bagchi [1] with $n = 49$ jobs on $m = 15$ machines. The considered objective functions of this instance are the minimization of the maximum completion time C_{max}, the minimization of the average completion time of all jobs $\frac{1}{n}C_{sum}$, and the minimization of the average tardiness of all jobs $\frac{1}{n}T_{sum}$.

Figure 3 gives a plot of the results in outcome space. The best solutions obtained with a multi-objective evolutionary algorithms using the fitness as-

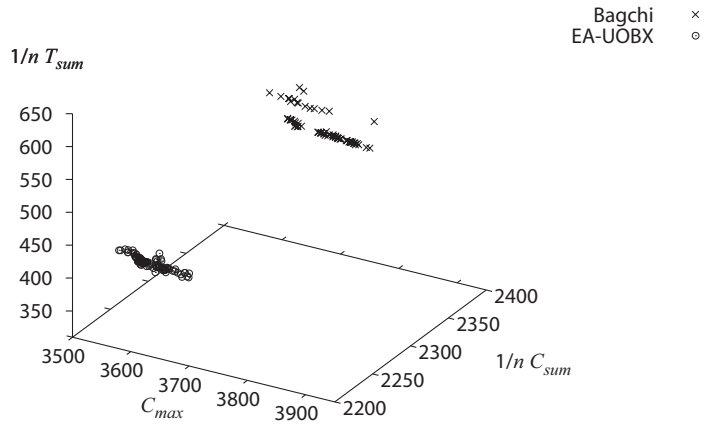


Fig. 3. Comparison of obtained approximation for the problem instance from [1] with previously known best solutions.

signment of Fonseca and Fleming [8] and a uniform order based crossover UOBX are compared with the results reported by Bagchi. It can be seen, that all alternatives of [1] are dominated. In particular with respect to the objective of minimizing the average tardiness of jobs, significant improvements have been obtained.

When closer investigating the results, it can be observed that the obtained alternatives are in rather close proximity to each other. The schedules share significant similarities both in outcome space, see Figure 3, and in alternative space. This indicates that Pareto optimal alternatives are closely concentrated in the search space of feasible alternatives $\mathcal{X}$ and helps to explain to some extend how metaheuristic search may work. As qualitatively good alternatives are typically close to other alternatives of high quality, this information may be exploited when computing neighboring alternatives using crossover or mutation operators.

4 Conclusions and discussion

A decision support system for multiple objective scheduling problems has been presented. It incorporates a set of metaheuristics that can be adapted to specific problem instances. As the user interface is highly visual, non-experienced users are able to solve scheduling problems under multiple objectives with comparably little knowledge.

Computational results have been gathered for benchmark instances taken from literature. It has been possible to observe the effectiveness of the implemented methods, even in comparison to the best known results of the test instances. While the results are satisfying with respect to that aspect, a further

development and improvement of the methods is unfortunately not permitted to the end user as the source code is not accessible.

After an approximation of Pareto-optimal alternatives has been obtained, an interactive decision making module based on the aspiration interactive method allows the identification of a most preferred schedule. The system may also be used to compare different approximation results of various metaheuristic approaches in terms of their approximation quality. It is therefore suitable for demonstrating the use, adaptation and effectiveness of metaheuristics to complex combinatorial optimization problems using the example of machine scheduling under multiple objectives.

The system successfully competed in the *European Academic Software Award*, a biannual contest of academic software in research and higher education. In this context, it has been evaluated by an international panel of experts. As it is aimed at end users who are not necessarily experts in the relevant field of metaheuristics or scheduling, its' realized concept differs from existing implementations. Rather than being generic like know software class libraries, it is specific. This bears the disadvantage of a potentially difficult adaptation to other problems than scheduling. On the other hand however, as it presents a closed system with no need of adapting and recompiling source code, it may also be used as a demonstration and learning tool in higher education. Based on the experiences gathered, we believe that it is able to stipulate the understanding and use of modern metaheuristics in research and higher education, and contribute to the further development and distribution of modern heuristics.

Acknowledgements

The author would like to thank Zsíros Ákos (University of Szeged, Hungary), Pedro Caicedo, Luca Di Gaspero (University of Udine, Italy), and Szymon Wilk (Poznan University of Technology, Poland) for providing multilingual versions of the software.

References

1. Tapan P. Bagchi. *Multiobjective scheduling by genetic algorithms.* Kluwer Academic Publishers, Boston, Dordrecht, London, 1999.
2. Matthieu Basseur, Franck Seynhaeve, and El-ghazali Talbi. Design of multi-objective evolutionary algorithms: Application to the flow-shop scheduling problem. In *Congress on Evolutionary Computation (CEC'2002)*, volume 2, pages 1151–1156, Piscataway, NJ, May 2002. IEEE Service Center.
3. J. E. Beasley. Obtaining test problems via internet. *Journal of Global Optimization*, 8:429–433, 1996.
4. J. Błażewicz, K. H. Ecker, E. Pesch, G. Schmidt, and J. Węglarz. *Scheduling Computer and Manufacturing Processes.* Springer Verlag, Berlin, Heidelberg, New York, 2. edition, 2001.

5. R. W. Conway, W. L. Maxwell, and L. W. Miller. *Theory of Scheduling*. Addison-Wesley, Reading, MA, 1967.
6. Richard L. Daniels. Incorporating preference information into multi-objective scheduling. *European Journal of Operational Research*, 77:272–286, 1994.
7. Richard L. Daniels and Joseph B. Mazzola. A tabu-search heuristic for the flexible-resource flow shop scheduling problem. *Annals of Operations Research*, 41:207–230, 1993.
8. Carlos M. Fonseca and Peter J. Fleming. Genetic algorithms for multiobjective optimization: Formulation, discussion and generalization. In Stephanie Forrest, editor, *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 416–423, San Mateo, CA, 1993. Morgan Kaufmann Publishers.
9. Tomáš Gál, Theodor J. Stewart, and Thomas Hanne, editors. *Multicriteria Decision Making: Advances in MCDM Models, Algorithms, Theory, and Applications*, volume 21 of *International Series in Operations Research & Management Science*. Kluwer Academic Publishers, Boston, Dordrecht, London, 1999.
10. Henry L. Gantt. Efficiency and democracy. *Transactions of the American Society of Mechanical Engineers*, 40:799–808, 1919.
11. B. Giffler and G. L. Thompson. Algorithms for solving production-scheduling problems. *Operations Research*, 8:487–503, 1960.
12. R. Haupt. A survey of priority rule-based scheduling. *Operations Research Spektrum*, 11(1):3–16, 1989.
13. Sang M. Lee and David L. Olson. Goal programming. In Gál et al. [9], chapter 8, pages 8.1–8.33.
14. V. Lotfi, T. J. Stewart, and S. Zionts. An aspiration-level interactive model for multiple criteria decision making. *Computers & Operations Research*, 19(7):671–681, 1992.
15. Michael Pinedo. *Planning and Scheduling in Manufacturing and Services*. Springer Verlag, Berlin, Heidelberg, New York, 2005.
16. Colin R. Reeves. Landscapes, operators and heuristic search. *Annals of Operations Research*, 86:473–490, 1999.
17. Eric Taillard. Benchmarks for basic scheduling problems. *European Journal of Operational Research*, 64:278–285, 1993.
18. Vincent T'kindt and Jean-Charles Billaut. *Multicriteria Scheduling: Theory, Models and Algorithms*. Springer Verlag, Berlin, Heidelberg, New York, 2002.
19. E. L. Ulungu, J. Teghem, P. H. Fortemps, and D. Tuyttens. MOSA method: A tool for solving multiobjective combinatorial optimization problems. *Journal of Multi-Criteria Decision Making*, 8:221–236, 1999.
20. David A. Van Veldhuizen and Gary B. Lamont. Multiobjective evolutionary algorithms: Analyzing the state-of-the-art. *Evolutionary Computation*, 8(2):125–147, 2000.
21. Philippe Vincke. *Multicriteria Decision-Aid*. John Wiley & Sons, Chichester, New York, Brisbane, Toronto, Singapore, 1992.
22. Darrell Whitley. Permutations. In Thomas Bäck, David B. Fogel, and Zbigniew Michalewicz, editors, *Handbook of Evolutionary Computation*, chapter C3.3.3, pages C3.3:14–C3.3.20. Institute of Physics Publishing, Bristol, 1997.
23. Andrzej P. Wierzbicki. Reference point approaches. In Gál et al. [9], chapter 9, pages 9.1–9.39.
24. James M. Wilson. Gantt charts: A centenary appreciation. *European Journal of Operational Research*, 149:430–437, 2003.