ABSTRACT

Title of Thesis: Computer Aided Design of Side Actions for Injection Molding of Complex Parts

Ashis Gopal Banerjee, Master of Science, 2006

Thesis directed by: Associate Professor Satyandra K. Gupta
Department of Mechanical Engineering

Often complex molded parts include undercuts, patches on the part boundaries that are not accessible along the main mold opening directions. Undercuts are molded by incorporating side actions in the molds. Side actions are mold pieces that are removed from the part using translation directions different than the main mold opening direction. However, side actions contribute to mold cost by resulting in an additional manufacturing and assembly cost as well as by increasing the molding cycle time. Therefore, generating shapes of side actions requires solving a complex geometric optimization problem. Different objective functions may be needed depending upon different molding scenarios (e.g., prototyping versus large production runs). Manually designing side actions is a challenging task and requires considerable expertise. Automated design of side actions will significantly reduce mold design lead times. This thesis describes algorithms for generating shapes of side actions to minimize a customizable molding cost function.

Given a set of undercut facets on a polyhedral part and the main parting direction, the approach works in the following manner. First, candidate retraction space is computed for every undercut facet. This space represents the candidate set of translation vectors that can be used by the side action to completely disengage from the undercut facet. As the next step, a discrete set of feasible, non-dominated retractions is generated. Then the undercut facets are grouped into undercut regions by performing state space search over such retractions. This search step is performed by minimizing the customizable molding cost function. After identifying the undercut regions that can share a side action, the shapes of individual side actions are computed.

The approach presented in this work leads to practically an optimal solution if every connected undercut region on the part requires three or fewer side actions. Results of computational experiments that have been conducted to assess the performance of the algorithms described in the thesis have also been presented. Computational results indicate that the algorithms have acceptable computational performance, are robust enough to handle complex part geometries, and are easy to implement. It is anticipated that the results shown here will provide the foundations for developing fully automated software for designing side actions in injection molding.

Computer Aided Design of Side Actions in Injection Molding of Complex Parts

by

Ashis Gopal Banerjee

Thesis submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Master of Science
2006

DEDICATION

To my parents and all those who made it possible

ACKNOWLEDGEMENTS

First and foremost I would like to express heartfelt appreciation and sincere gratitude to Dr. Satyandra K. Gupta for providing me this wonderful opportunity to conduct research under his astute guidance. His boundless energy, wonderful analytical skills, cool and calm composure and motivational power has made this experience an invaluable one. I am sure this will stand me in good stead in my future research and professional career as well. I would also like to thank Prof. Abhijit Dasgupta and Prof. Shapour Azarm for readily accepting to be in my thesis committee.

Then I am thankful to University of Maryland, College Park, Department of Mechanical Engineering and The Institute for Systems Research for their support and facilities. I also want to express my appreciation to all my CIM lab friends, namely, Alok, Abhijit, Tao, Antonio and Xin for their help. Special thanks go to Alok for his invaluable advice on C++ and geometric kernel programming and to Abhijit for proof-reading my thesis and providing constant words of encouragement.

Of course, all this has been made possible by my parents and other family members. I can never thank Mom and Dad enough; they are the most wonderful parents one can ever hope to have. Their unfailing emotional support, love and affection have carried me through trying circumstances. I would not have been able to enjoy this achievement fully without them.

Last but not the least I want to thank all my dear friends: Arvind, Max, Mukul, Basu, Sahoo, Vipul, Rajat, Singhal, Ma, Srini, Avik, Sourav, Jignesh, Ramakanth, Milind, Tejas, Srilatha, Shivani, Raee, Madhura and many others for advice, friendly rebukes and loads of affection.

TABLE OF CONTENTS

List of Figures

List of Tables

Chapter 1

INTRODUCTION

This Chapter is arranged in the following manner. Section 1.1 gives an overview of injection molding process, Section 1.2 describes single material injection molding process is greater details, Section 1.3 presents the motivation behind the research undertaken in this thesis, Section 1.4 briefly discusses the research issues and Section 1.5 describes the outline of this thesis.

1.1 Injection Molding Background

Injection molding (IM) is one of the most widely used plastic manufacturing processes nowadays [Fu99b]. It is a near net-shape manufacturing process that can produce parts having good surface quality and accuracy. Moreover, this process is suitable for mass volume production due to fast cycle time. It combines the functionality of multiple parts into one, thereby enabling production of complex designed parts. Typical examples of products made by injection molding include appliance casings like computer monitors, CPUs, aircraft and automobile parts, utensils, toys and so on.

Broadly, injection molding can be classified into two types: single material molding (SMM) and multi material molding (MMM). In essence, the SMM process is simple: plastic is melted and injected into a tool called *mold* where it then cools and re-hardens to take the shape of the mold cavity. The cooled *part* can then be ejected, and the cycle repeats. But before going into the details of this process in the next Section, let us briefly discuss MMM.

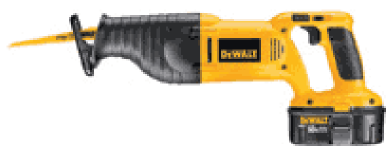
Over the last few years, a wide variety of multi material injection molding processes have emerged for making multi material objects, which refer to the class of objects in which different portions are made of different materials. Due to fabrication and assembly steps being performed inside the molds, molded multi material objects allow significant reduction in assembly operations and production cycle times. Furthermore, the product quality can be improved, and the possibility of manufacturing defects, and total manufacturing costs can be reduced. In MMM, multiple different materials are injected into a multi-stage mold. The sections of the mold that are not to be filled during a molding stage are temporally blocked. After the first injected material sets, then one or more blocked portions of the mold are opened and the next material is injected. This process continues until the required multi-material part is created. Nowadays, virtually every industry (e.g., automotive, consumer goods, toys, electronics, power tools, appliances) that makes use of traditional SMM process is beginning to use multi material molding processes. Some common applications include multi-color objects, skin-core arrangements, in-mold assembled objects, soft-touch components (with rigid substrate parts) and selective compliance objects. Typical examples of each class of application are shown in Figure 1.1.

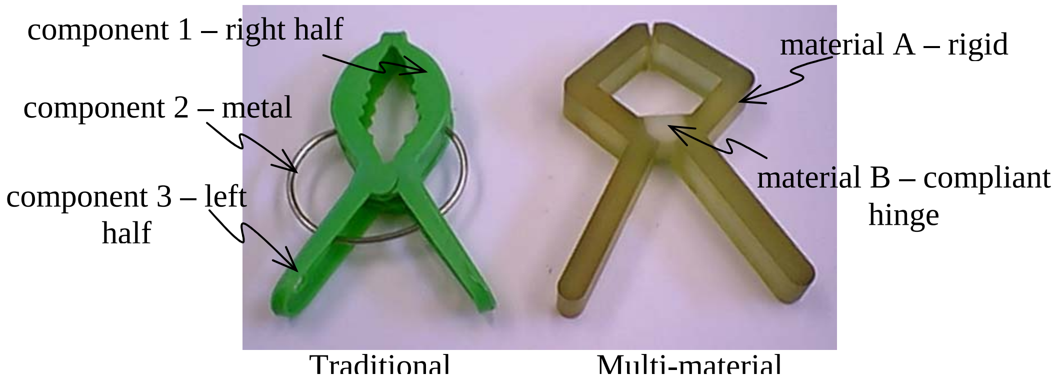
There are fundamentally three different types of multi material molding processes. Multi-component injection molding is perhaps the simplest and most common form of MMM. It involves either simultaneous or sequential injection of two different materials through either the same or different gate locations in a single mold. Multi-shot injection molding (MSM) is the most complex and versatile of the MMM processes. It involves injecting the different materials into the mold in a specified sequence, where the mold cavity geometry may partially or completely change between sequences. Over-molding simply involves molding a resin around a previously-made injection-molded plastic part. Each of the three classes of MMM is considerably different. Each specific MMM process requires its own set of specialized equipment; however, there are certain equipment requirements that are generally the same for all types of MMM.

(a) – Taillight (multi-color MMO)

(b) – Assorted brushes (skin/core arrangement)

(c) – One-piece syringe (in-mold assembly)


(d) – Cordless saw housing (soft-touch grip)



(e) Compliance clips

Figure 1.1: Different multi material molded objects

1.2 Single Material Molding

Although the concept behind single material molding is simple, the actual process must be carefully controlled to produce acceptable parts. Additionally, expensive and sophisticated equipment is required to produce even the simplest of parts. The basic equipment required is discussed below.

1.2.1 Single Material Molding Equipment

SMM is carried out on a molding machine. There are many different types of specialized molding machines, but they all share the same four basic components: 1) an injection unit, 2) a clamping unit, 3) a mold, and 4) a controller [Bryc96]. A schematic of a generalized SMM machine is illustrated in Figure 1.2, and a photo of an industrial SMM machine is shown in Figure 1.3. Note that there are two platens on a molding machine - one moving, and one stationary. These plates are located on the clamping side and injection side of the machine, respectively.

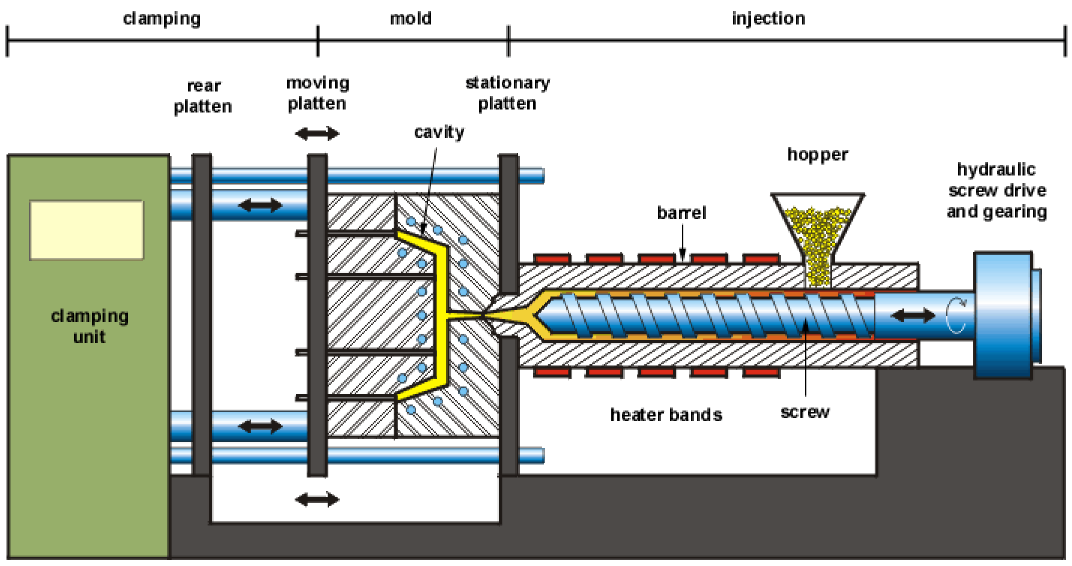


Figure 1.2: Schematic of a typical injection molding machine
Source: http://www.idsa-mp.org/proc/plastic/injection/injection_process.htm

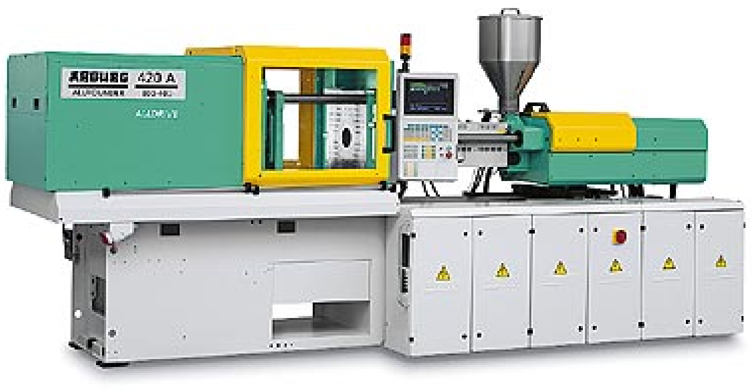


Figure 1.3: Photo of a typical injection molding machine
Source: http://www.arburg.com/english/products/po_a0001.htm

- Injection unit

The purpose of the injection unit is to force or inject the molten plastic into the mold under pressure. The injection unit consists of a plastic-feeding hopper, a heated barrel (heating cylinder), an injector, and an outlet nozzle. The injector is usually a reciprocating screw, but can also be a simpler hydraulic plunger-type device [Bryc96]. All of these components work in conjunction to feed the plastic, melt it, and then inject it into the mold. There are many different injection unit configurations available, based on the needs of the particular molding job.

- Clamping unit

The clamping unit works to hold the mold closed during injection and actuates the ejector when it opens. The clamping mechanism is either a hydraulic press or a mechanical lever system and must be strong enough to resist the large forces generated by the injection pressure on the mold walls. Additionally, the clamping unit must incorporate some sort of ejector device to facilitate part removal from the mold [Bryc96].

- Mold

The mold is a complex tool that contains the cavity or cavities in which the plastic parts will cool. The present work will mainly deal with molds and ignore the other components of the SMM system. There are several different types of molds, including two-piece molds, three-piece molds and multi-piece molds (space puzzle molds). Two-piece molds have only one primary parting surface and consist of two main mold pieces which are separated along a *main mold opening direction* (also known as parting direction) to eject the solidified part. On the other hand, multi-piece molds have many parting directions, more than one primary parting surface and consist of multiple main mold pieces. They can be visualized as 3D jigsaw puzzles where all the mold pieces fit together to form the mold cavity and then can be disassembled to eject the molded part [Priy02]. However, two-piece molds are used most commonly in industry and will be discussed in details hereafter.

Essentially, two-piece molds consist of a cavity located inside a set of plates divided into two halves: 1) a moving half, and 2) a stationary half [Boot02]. The stationary half (*cavity*) is located on the injection side of the molding machine, bolted onto the stationary platen and connected to the injection unit. It is typically used to create the external faces of the part. The moving half of the mold (*core*) is bolted onto the clamping unit and moves with it during the mold opening and closing phases. Core creates the internal faces of the part. A highly simplified schematic of a two-plate mold and the part it would produce is illustrated in Figure 1.4.

In reality, the mold is not simply two halves with a single cavity. Rather, it is an intricate system of moving plates with a resin flow system (gates, runners, and a sprue) that feeds multiple cavities. Additionally, molds must have a network of hydraulic cooling lines as well as some type of part ejection system. Figure 1.5 schematically illustrates a two-cavity, two-piece mold in greater detail than Figure 1.4. Note that the only components of the mold that actually contain the geometry of the molded part are the core and cavity. The rest of the components serve as a support infrastructure for these pieces and are collectively referred to as the mold base. In many cases, it is possible to produce a family of different (but similarly-sized) parts on the same mold base by using different cores and cavities [Fowl04].

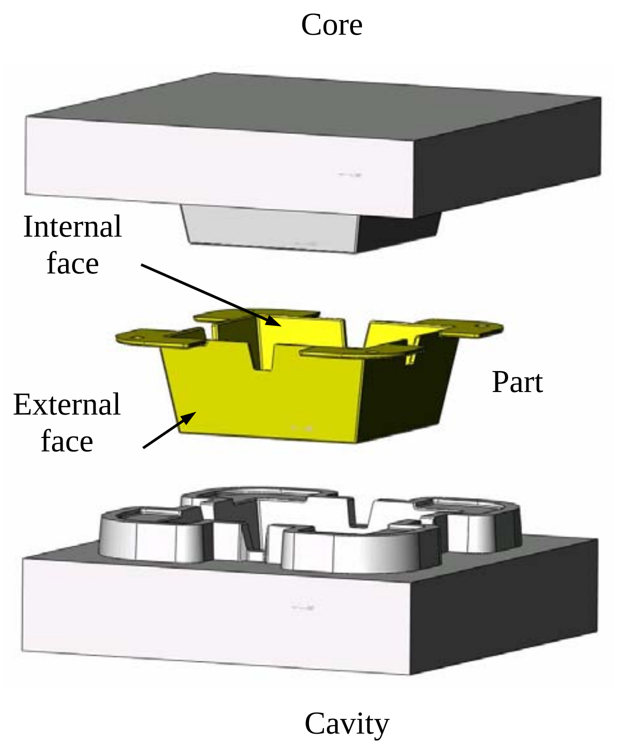


Figure 1.4: Part molded by a two-piece mold

Several types of mold subsystems are sometimes employed in special molding situations. While not required for basic two-piece single material (SM) injection molding, they may serve to increase part quality, reduce costs, and/or produce features and geometries not normally realizable with standard equipment. Normally these devices must be custom-built directly into the mold base. Two common subsystems are described below.

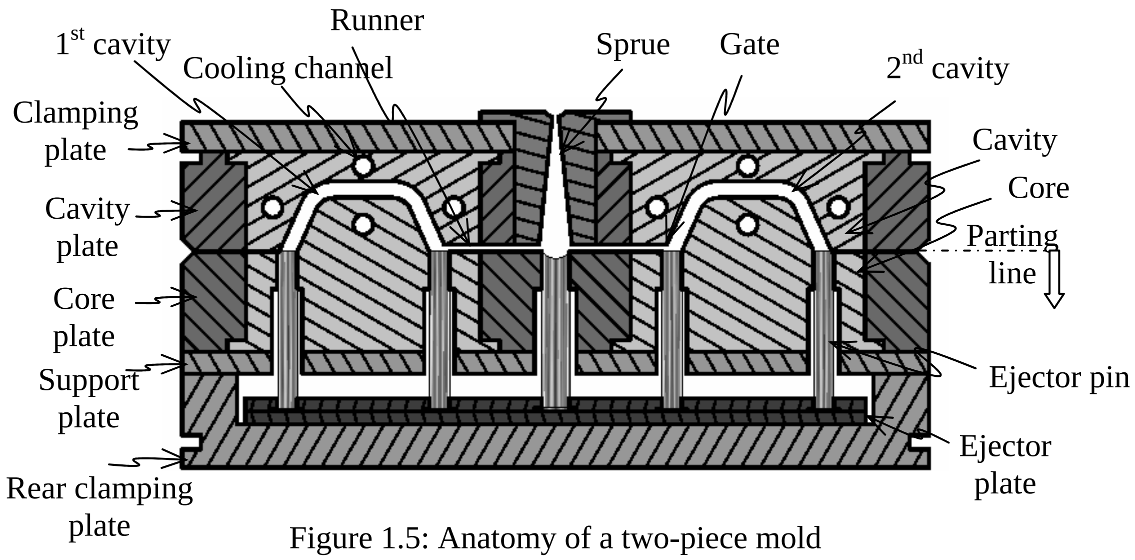


Figure 1.5: Anatomy of a two-piece mold

A very common type of optional mold subsystem is called a *side action*. Side actions are secondary mold pieces (core and cavity form the main mold pieces) which allow *undercuts* to be molded into the part. Undercuts are patches or regions on the part boundaries that are not accessible along the main mold opening directions. They prevent standard two-piece molds from opening after the molten plastic has solidified. For example, any kind of external depression or hole parallel to the parting surface is a type of undercut. Hence the most important point to be noted here is that side actions are removed from the part using translation directions different than the main mold opening direction. Use of side actions is illustrated in Figure 1.6.

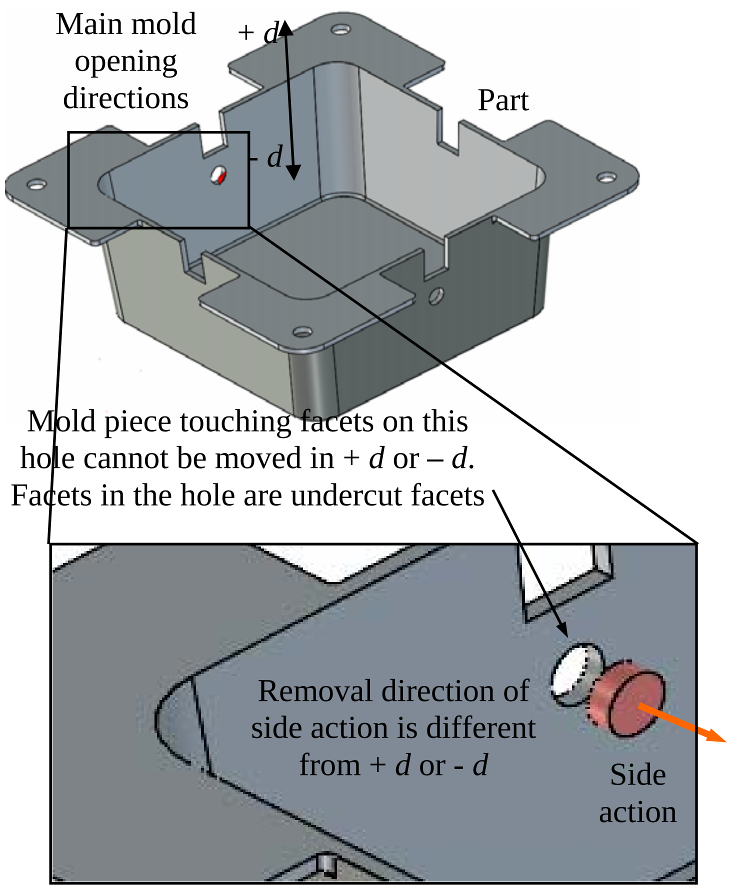

Main mold opening directions
+ d
- d
Part
Mold piece touching facets on this hole cannot be moved in + d or – d. Facets in the hole are undercut facets
Removal direction of side action is different from + d or - d
Side action

Figure 1.6: Side action removal in a direction different from the main mold opening direction

Now side actions are typically actuated by some sort of sliding mechanisms. Figures 1.7a and 1.7b illustrate an undercut created by part geometry, and Figures 1.7c and 1.7d illustrate one way to produce undercuts without mold locking. After the part cools, the cam in Figure 1.7d is shifted to the right, allowing the mold to be opened. The sideways movement of the slide can be actuated by one of several different kinds of mechanism including a side-pull pin, or a hydraulic piston. It should be noted that such devices must be specifically made for each case, and can significantly drive up the cost of the mold base [Fowl04]. The current work specifically deals with automated design of side actions for complex industrial parts so as to minimize the overall molding cost without taking into account the actual mechanism design.

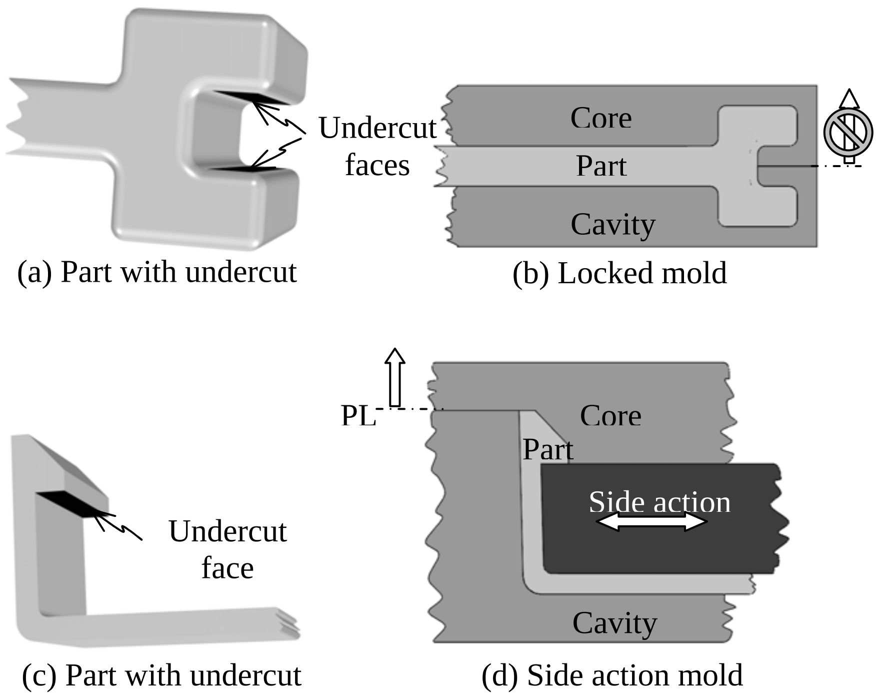
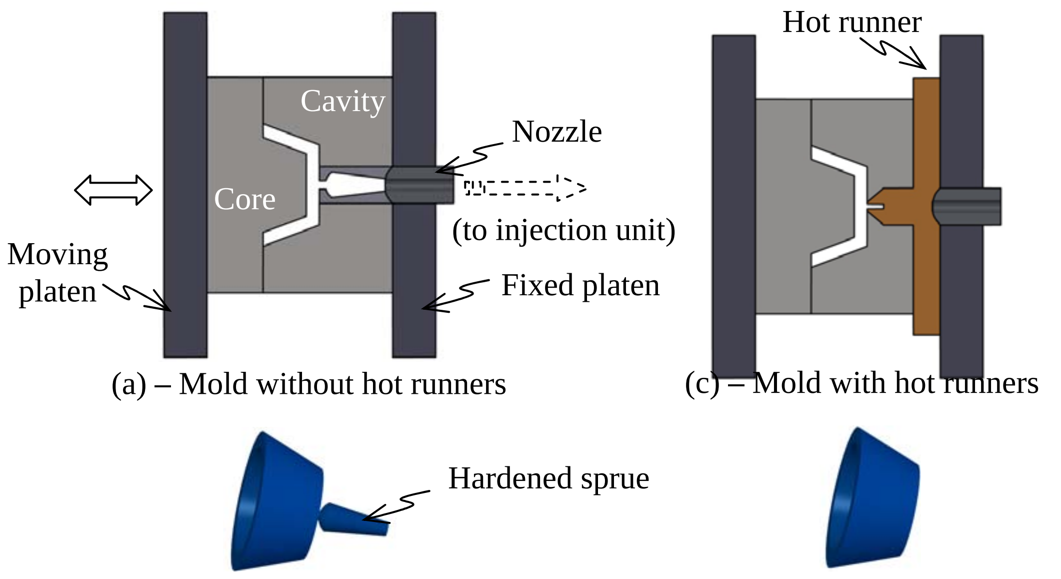


Figure 1.7: Molds with and without side action

Another common type of mold subsystem is called a hot runner system. These devices extend the nozzle of the injection unit directly into the cavity gate by keeping the resin molten through a system of heated channels. This eliminates the undesirable solidification of the sprue and runners, which are then ejected with the part (and in many cases still attached to it). This cuts down on resin scrap, manual runner-removal operations, and generally improves the resin flow characteristics inside the mold. Figure 1.8 illustrates two molding scenarios: one without a hot runner system and one with. The resulting parts are shown below their respective mold configurations. The key difference to note is that the part in Figure 1.8b has the hardened sprue attached to it while the part in Figure 1.8d does not. It would have to be later removed through grinding or some other method. The removed sprues would then either be discarded or recycled into resin pellets and re-used. As with side actions, hot runner systems must be custom-built into each mold and drive up the total cost of the tool. Figure 1.9 shows photos of typical hot runner systems.

- Controller

In order to operate the various machine components all molding machines must include a controller unit. The controller is a sophisticated programmable computer, responsible for a host of tasks. This includes: attaining and holding the desired process variables (e.g. injection pressure and temperature), controlling the cooling system, and actuating the clamp mechanism and any mold subsystems. It must be able to accurately control the process temperatures and time the various actions in order to produce decent molded parts. Typically, the controller is fully integrated with the molding press and can be specially configured to meet the specific needs of the molding job. Machine manufacturers are able to provide custom SMM machines with whatever combination of injection units, clamping units, controllers, and auxiliary equipment necessary for most particular molding applications.



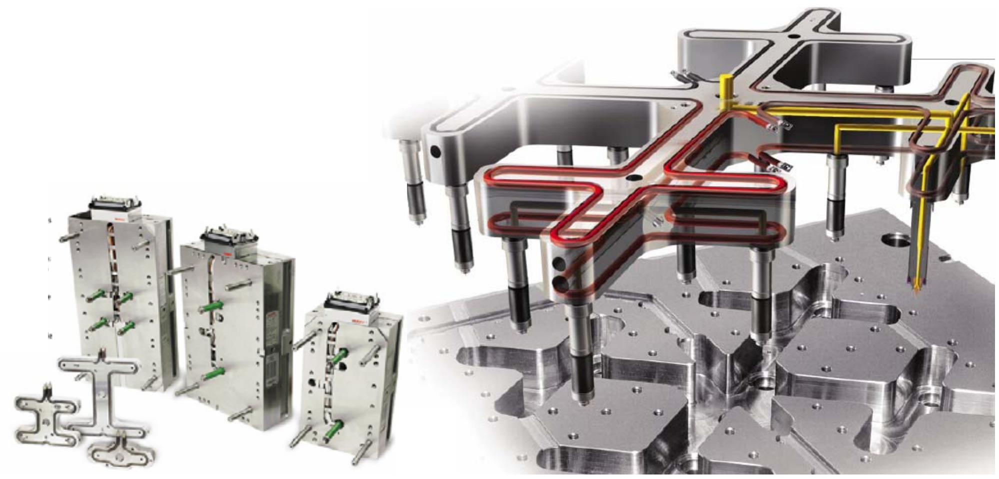
(a) – Mold without hot runners

(c) – Mold with hot runners

(b) – Resulting part from (a)

(d) – Resulting part from (c)

Figure 1.8: Schematic of molds with and without hot runners



(a) – Typical hot runner components

(b) – Detail of hot runners

Figure 1.9: Photographs of hot runner systems
Source: Husky Inc. product brochure

1.2.2 Single Material Molding Process

The basic two-piece SMM process without presence of any side action, as illustrated in Figure 1.10, is outlined as follows:

1) **Closing the mold:** The mold closes so the cycle can begin.

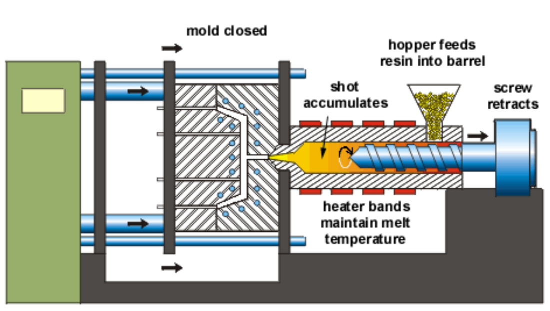
2) **Plasticizing the resin:** (Figure 1.10a) The hopper feeds solid pellets or grains of the plastic resin into the barrel where it becomes molten due to the heating bands and friction caused by the rotating screw. The molten plastic accumulates at the front of the barrel (the nozzle side) as the screw retracts to the rear of the barrel.

3) **Injecting the resin:** (Figure 1.10b) When enough molten plastic has accumulated for a full shot, a valve in the nozzle is opened and the screw rapidly advances forward, quickly injecting the plastic into the mold cavity.

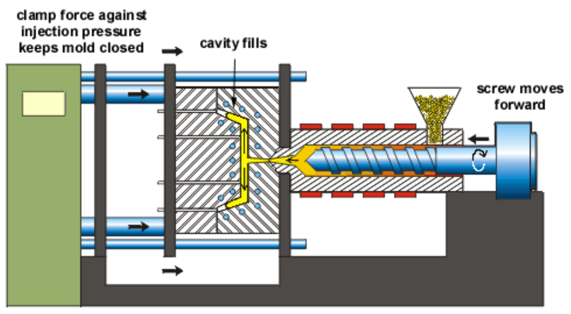
4) **Cooling the part:** (Figure 1.10c) The screw continues to push plastic into the mold in order to create a holding pressure. This ensures adequate filling. As this happens, the molten plastic begins to cool and solidify toward the inside. This natural cooling is expedited by convection due to coolant flowing through channels inside the mold.

5) **Ejecting the part:** (Figure 1.10d) After adequate cooling time has elapsed, the mold is opened. Some sort of ejector device is actuated in this process and the part is ejected from the mold and collected. After this, the cycle repeats from step 1.

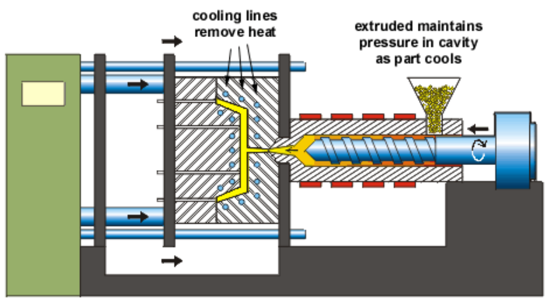
However, as shown in Figure 1.11, certain additional steps are required after part solidification if side actions are included. First, side actions retract (translate) as shown in Step (2). Then the part needs to be taken out along with the core without intersecting the retracted side actions. Since Step (3) involves a relative motion between the part and the side actions, this can also be modeled as the second translational motion that needs to be imparted to the side actions for complete disassembly. Step (4) is same as stage 5) mentioned above and then onwards the previous cycle is followed till we again come to part cooling.


mold closed
hopper feeds resin into barrel
shot accumulates
screw retracts
heater bands maintain melt temperature

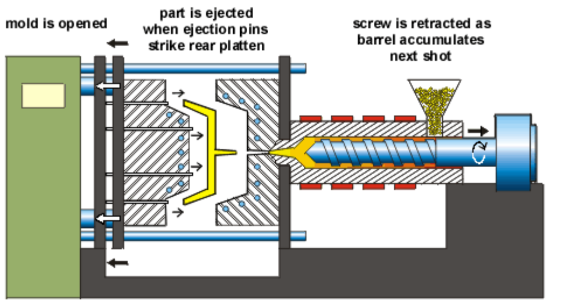
(a) – Step 2: Plasticizing



(b) – Step 3: Injecting


cooling lines remove heat
extruded maintains pressure in cavity as part cools

(c) – Step 3: Cooling


mold is opened
part is ejected when ejection pins strike rear platten
screw is retracted as barrel accumulates next shot

(d) – Step 4: Ejecting

Figure 1.10: Schematic of the injection molding process
Source: Hhttp://www.idsa-mp.org/proc/plastic/injection/injection_process.htmH

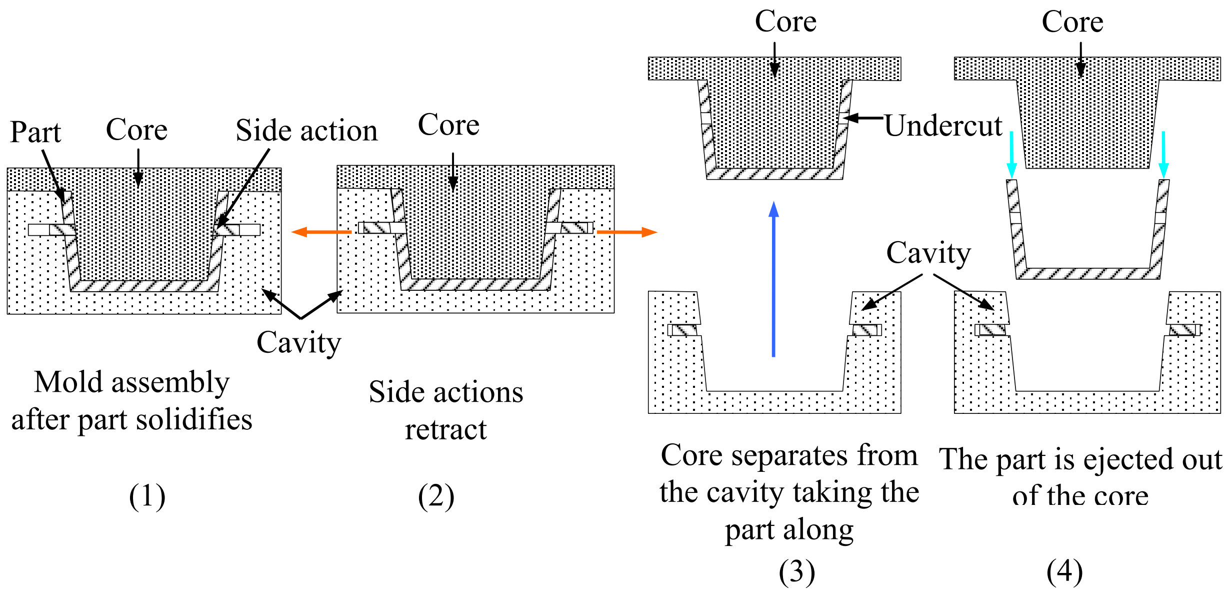

Part
Core
Side action
Core
Core
Core
Undercut
Cavity
Cavity
Mold assembly after part solidifies
(1)
Side actions retract
(2)
Core separates from the cavity taking the part along
(3)
The part is ejected out of the core
(4)

Figure 1.11: Steps in injection molding after part solidification

1.3 Motivation

Automated mold design: In the field of manufacturing, productivity is achieved by transforming a product from the conceptual design stage to finished good in the market quickly and inexpensively. Due to increased competition in today's global economy, shorter design and manufacturing lead times, enhanced quality and reliability and frequent design modifications are often desired. Typically, design phase is the bottleneck in any product development process since it is a complex and time-consuming task [Priy02]. Hence we need fast computers coupled with efficient algorithms to generate designs automatically in a very short period of time.

Design changes become more costly at the later stages of production. So it is important to detect any design errors and inconsistencies at an early stage. This can now be done easily with the help of virtual prototyping; automating the design process is a pre-requisite for that.

However, the importance of design automation does not end here. Let us consider a simple scenario where a company has been asked to design the entire mold system for a new product. The traditional approach is to the use the expertise of an experienced mold designer. Since the entire process will be done manually, it will take weeks to complete the design. The designer may not consider all possible designs. Moreover, the generated design may not be the optimum one. In addition, as there is no effective communication between the part designer and the mold designer, usually multiple iterations are required till the part and mold designs are finalized. Sometimes, it becomes very difficult even for a highly skilled designer to visualize the mold geometries for complex parts involving multiple undercuts that need to be split into smaller moldable regions. Keeping in view all these issues, automated mold design is no longer a nice-to-have optional feature; it is a necessity.

Although almost all the CAD systems such as ProEngineer, Unigraphics, Solid Edge, SolidWorks etc. provide mold design software packages, none of them can satisfactorily design all the mold pieces for injection moldable plastic parts fully automatically without any form of manual intervention. Given a parting direction, they may be able to design the core and cavity; but designing side actions and that too, optimally, is beyond their current capabilities. However, as discussed in the previous Section, side actions constitute an important part of the mold system. They contribute significantly to the overall molding cost by increasing the molding cycle time (due to actuation) and the mold manufacturing cost. Hence, it will be useful to have a system that can automatically accomplish this task for us. It must be capable of designing shapes of side action solids automatically from the geometrical information present in undercuts, without any inputs from the user. At the same time, by changing the operational parameters, the user must be able to obtain alternative designs which will satisfy his specific needs. In other words, the system should be flexible or customizable enough to allow the user to explore various design options and select the optimum one. This will also provide the user with a reasonable estimate of the final cost that will be incurred during the actual operational phase which will be beneficial to both the design and manufacturing groups.

Another important application is in automated generation of feasible molding sequences in multi material molding [Bane06b]. Infeasibility of molding sequences implies that the multi-material object cannot be manufactured because the proposed molding sequence is infeasible. Usually this problem is caused by the presence of deep undercut features and it manifests itself in different ways for different multi-material molding processes. The three common reasons are listed as follows.

- The two components are made of different materials and one of them has a higher melting point than the other. Then this component has to be molded first and if it is non-moldable then multi material molding cannot be carried out. This scenario has been illustrated in Figure 1.12. The moldability of the other component and the gross object is immaterial in this case.
- Let us now consider that the two components are made of same material (different colors) or different materials having comparable melting points. If none of the two components are separately moldable, then no feasible sequence exists as the molding process cannot be started at all.
- If the faces that need to be demolded in the second stage create impossible to mold undercuts multi material molding is not possible.

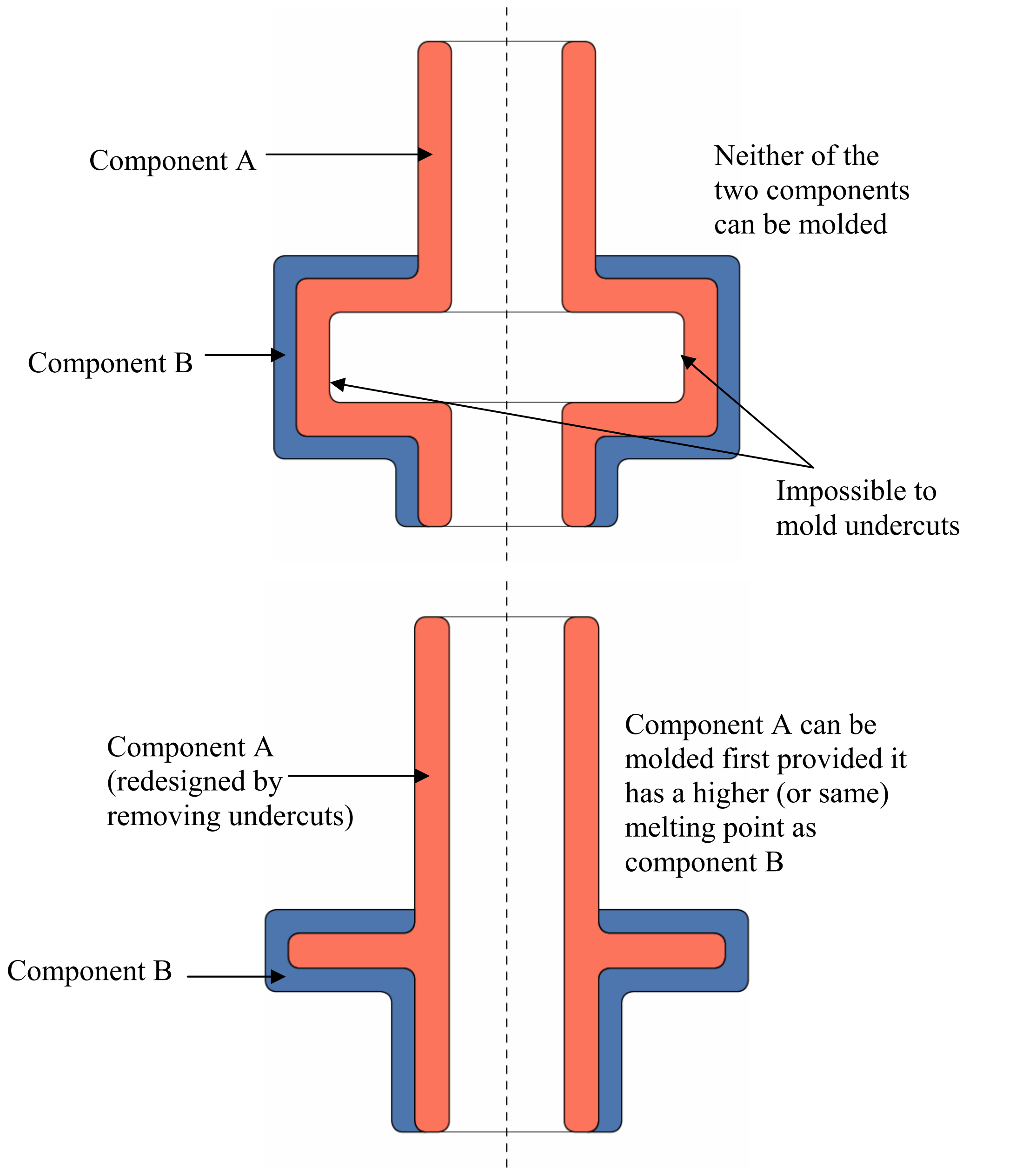

Component A
Neither of the two components can be molded
Component B
Impossible to mold undercuts
Component A (redesigned by removing undercuts)
Component A can be molded first provided it has a higher (or same) melting point as component B
Component B

Figure 1.12: Redesigning a component after identifying impossible to mold undercuts to create a feasible molding sequence

1.4 Research Issues

There are two key research issues for which efficient solutions need to be obtained in order to automate the process of designing side actions for complex industrial parts. They are discussed below.

1. *Computing feasible retraction spaces*: From the discussion in Sub-section 1.2.1 it is clear that side actions need to retract after the molten plastic solidifies in the mold cavity. Hence, sufficient space must be available in the *mold free space* (regularized Boolean difference between a rectangular bounding box and the part) to move the side actions along a direction different from the main mold opening direction and then to remove the part along with the core without resulting in any collision(s). The difference between regularized and non-regularized Boolean difference has been explained in Appendix A. In other words, the space available for intersection-free (feasible) retraction of side actions has to be computed in order to determine their shapes. So one can think of computing the feasible retraction spaces for all the *undercut facets* individually. By undercut facets, one means the triangular faces representing the undercut region that will be eventually molded by the side actions. A CAD model and its faceted representation are shown in Figure 1.13 below. The feasible retraction space for a particular undercut facet *f* is shown in Figure 1.14. It will be defined in a more mathematically rigorous manner in Section 3.4. However, the point to be noted here is that computing these spaces precisely and efficiently is a challenging task. Firstly, all other facets on the part that may potentially block free translations of the undercut facets need to be identified. Then, an approach similar to that adopted in motion planning of polyhedral robot in an environment occupied by many other polyhedral obstacles may be utilized to solve this problem. Although this is a well-known technique, existing methods try to avoid such computationally expensive procedure by using ray-accessibility or local concave region based

techniques. Since they often do not yield satisfactory results, a way for computing feasible retraction spaces has to be found out.

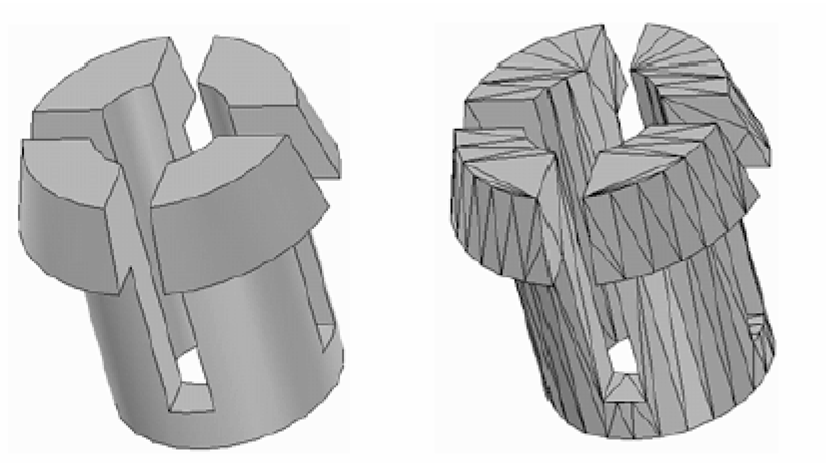


Figure 1.13: Faceting (from [Priy02])

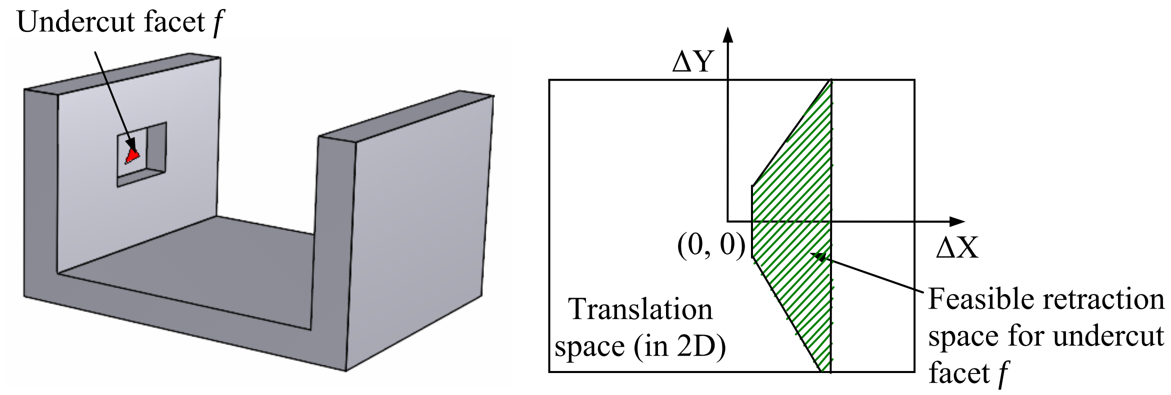

Undercut facet *f*
ΔY
(0, 0)
ΔX
Translation space (in 2D)
Feasible retraction space for undercut facet *f*

Figure 1.14: Feasible retraction space

2. *Forming optimum (or near-optimum) set of side actions*: Once all the feasible retraction spaces are obtained, the next challenge is to obtain an optimum solution. Here optimum solution refers to a particular design of a set of side actions such that the molding cost is minimized and all of them have non-null feasible retraction spaces associated with them. So one first needs to frame an objective (cost) function using

geometrical parameters only. Then, one has to search for the optimum solution within the feasible domain, i.e. retraction spaces. Since every undercut facet has a feasible retraction space, the trick lies in combining them suitably so that many can be handled by a single side action and the amount of retraction needed is as small as possible. These are conflicting objectives and hence one is confronted by a combinatorial geometric optimization problem. An intelligent heuristics based search strategy needs to be devised to solve this problem in a reasonable amount of time.

1.5 Thesis Outline

The remaining part of the thesis is organized as follows. Chapter 2 describes the published literature concerning automated mold design. Chapter 3 presents the problem formulation, establishes the theoretical foundations behind the proposed technique and gives an overview of it. Chapter 4 describes the first major step of the proposed approach in details. Chapter 5 explains the remaining two major steps and presents implementation results. Most of the material covered in Chapters 3, 4 and 5 has been already published in [Bane06a]. Finally, Chapter 6 summarizes the conclusions reached from this research, highlights the anticipated benefits and provides suggestions for future extensions of this work.

Chapter 2

LITERATURE SURVEY

This chapter provides a survey of the state of the art in the field of automated mold design. However, attention will be restricted to design of mold pieces only. Design of sprue, runner and ejection systems are not discussed in this chapter. This is because the geometric aspect of side action design is almost entirely related to the design of mold pieces only. The arrangement of cooling and ejections systems can be analyzed by modeling the fluid flow and the resultant temperature and stress distribution using numerical techniques such as finite element analysis (FEA). This chapter is organized as follows. Section 2.1 reviews all the existing approaches in determining parting directions for a given plastic part. Section 2.2 surveys the popular techniques in determining parting line and parting surface. Once all these three geometric entities have been determined, it is possible to design the main mold pieces (core and cavity) for a 2-piece injection molding system. Section 2.3 then reviews the available methods for recognizing undercut features. Once such features have been identified, one can go ahead and actually design the side cores. Existing approaches to do that have been surveyed in Section 2.4. Section 2.5 reviews the work done in other forms of injection molding, namely multi-piece and multi-stage or multi-material molding. Section 2.6 reviews some of the work done in related fields such as die design for other types of plastic processing technologies. Finally, Section 2.7 summarizes the findings and highlights the need for new algorithms to tackle the problems not adequately addressed by the existing approaches.

2.1 Determination of Parting Directions

Significant progress has been made in the area of computing main parting direction. Pioneering work was done by Chen et al. [Chen95]. They first linked demoldability (i.e. the problem of ejecting all the mold pieces from the mold assembly such that they do not collide with each other and with the molded part) to a problem of partial and complete visibility as well as global and local interference. If a surface is not completely visible, then either it is blocked locally by parts of the same surface, or globally by other surfaces. They reduced global accessibility of undercuts (or sealed pockets that are the non-convex regions of a part) to a local problem and used the intersection of the visibility maps of all the pocket surfaces to find a plausible parting direction. Basically, a visibility map or VMap is the spherically convex region on the Gauss sphere on which any point corresponds to a direction from which the entire surface is locally visible. The sealed pockets are obtained as the regularized Boolean difference between the convex hull of an object and the object itself.

Now as the set of directions from which a pocket surface is completely visible and thus demoldable is the VMap of the surface, the problem of finding an optimal pair of parting directions basically boils down to one of locating a pair of antipodal points such that they are contained in (or covered by) maximum number of pocket surface VMaps. All the other pocket surfaces corresponding to VMaps not covered by the pair of points will require side cores. The authors have proposed an O(nm log m) running time algorithm to solve this problem by traversing through the cells using certain adjacency relationships, where n is the number of faces in the polyhedral object and m is the number of spherical polygons. Mathematical definition of O notation has been explained in Appendix B.

However, this approach suffers from few limitations. One such limitation is that it assumes that a side core is used for a pocket which has an empty VMap and is thus not completely visible from any direction. However, the use of cores can be avoided by sub-dividing a pocket based upon the notion of partial visibility. Chen et al. [Chen95] precisely address this issue in their subsequent work. The main hypothesis behind this paper is that if a surface is not completely visible and hence cannot be separated from the mold assembly along any direction, it can be decomposed into two portions: those that are separable and those that form undercuts. The separable portions are known as visibility polyhedra (VP).

Now to subdivide such a surface, directions in which it is partially visible need to be considered. The directions along which a surface is partially visible are not equivalent because different portions of the surface are visible or invisible in different directions. In order to minimize the number of undercuts, the spherical polygon representing the viewing directions along which a surface is partially visible needs to be further partitioned into smaller regions. The spherical map thus obtained is called the augmented visibility map (AVM) of the surface.

The authors have presented an O(n^2) time algorithm known as the pocket decomposition algorithm to partition a sealed pocket into a set of visibility polyhedra and undercuts, where n is the number of faces in the pocket. To compute the AVM, firstly the set of directions in which a sealed pocked S is partially visible (known as the range of partial visibility PV(S)) is computed in O(m log m) time, where m is the number of lid faces in *S*. Now appearance, disappearance and merging of undercuts are associated with certain topological changes in the walls constructed to form a VP. Only one type of critical event occurs- namely the projection of a vertex lies on the projection of a non-adjacent edge (VE). Since the occurrence of VE event is represented by $O(n^2)$ great circles, these $O(n^2)$ loci of critical directions partition the unit sphere into $O(n^4)$ regions. Each such partitioned region is then traversed and the pocket decomposition algorithm is applied to determine the number of undercuts for any arbitrary viewing direction in each region. Thus, overall the algorithm has an $O(n^6)$ running time.

Hui and Tan [Hui92] heuristically generated a set of candidate parting directions that consisted of planar face normals and axes of cylindrical faces. Based on the observation that the face normals of the openings of the cavity solids (pockets in [Chen93]) also define a zone of possible directions for clearing the corresponding undercut, Hui [96] added these to the set of candidate parting directions. He also developed a partitioning scheme to sub-divide the pockets without destroying global accessibility. For a candidate parting direction $\vec{d}$, a pocket is partitioned into by a series of planes each containing an edge of the pocket and a vector parallel to $\vec{d}$. The elements obtained after partitioning are convex and are either completely blocked or free in $\vec{d}$. Each candidate parting direction is then evaluated and an optimum direction requiring minimum number of side and split cores is selected as the main mold opening direction. The primary limitation of this approach is that the heuristically found set of parting directions may not be complete.

Chen [Chen97] has proposed an entirely different approach to determine the most promising parting direction based on fuzzy logics. For a given 3D CAD model, this method first finds the minimum volume bounding box of the model. Then three pairs of possible parting directions are defined along the surface normals of the bounding box. Multiple criteria such as number of undercuts, draw depth, projected area and other designer preferences are encoded as fuzzy weighing factors for evaluating the optimal parting direction. Uncertainty inherently present in the assessment of the weights is accounted for by the suggested fuzzy heuristic. Although the technique is novel, only three different parting directions are compared here; this significantly restricts its practical usability.

Dhaliwal et al. [Dhal03] have described algorithms (with exact mathematical conditions) for computing global accessibility cones for each face of a polyhedral object. Their work is significantly different from that done by Chen et al. because that have no longer restricted themselves to deal with local interference. The accessibility cones are represented in the form of a matrix so that useful information can be retrieved by queries easily later on. The boundary of the unit sphere is partitioned into a finite number of spherical triangles, each representing a set of directions. Rows of the matrix represent spherical triangles, whereas columns represent the planar, triangular facets. Each entry in the matrix indicates whether a facet is completely accessible from a spherical triangle or not. Since the global accessibility cone of a convex-hull facet is always a hemisphere, only non-convex-hull facets need to be considered.

Another interesting aspect of their work is that they have proposed two alternative algorithms- one exact and the other approximate to generate the overall accessibility cone matrix. The approximate scheme has been devised to avoid the numerical accuracy problems that inevitably creep in any geometric computations, destroying their robustness. This approximation works fine since it errs on the safer side and provides a more conservative result. Thus, this work not only improves upon the interference scheme originally proposed by Chen et al., it gives provably sound as well implementation friendly algorithms.

Rappaport and Rosenbloom [Rapp94] first introduced the concepts of moldability and castability of simple polygons and related moldability to monotonicity. After determining all the forward maximal monotone chains of a simple polygon, they have utilized that to develop a simple optimal algorithm to determine 2-moldabililty in linear time and 2-castability in linear-logarithmic time. Although they have only considered the problem of 2-moldability in 2 dimensions (planes), their work is ground breaking in the sense that they have used principles of computational geometry extensively for the first time in such a real-world engineering application to ensure disassemblability of mold pieces.

Inspired by this, a generalized and immensely significant result has been established by Ahn et al. [Ahn02]. They have given a comprehensive algorithm to test whether, given a mold and two opposite directions, the mold can be partitioned into two parts such that they can be removed along those two directions, without colliding with the object or the other mold part. Moreover, they have described an algorithm to find all such possible directions for polyhedral objects in polynomial time. Using a number of lemma and theorems dealing with the mathematical concepts of monotonicity, silhouettes and shadow curves, they have given necessary and sufficient conditions for both the problems. Some of the major contributions of this work can be summarized as follows:

- An object is moldable if and only if it is vertically monotone.

- An object is vertically monotone, and therefore moldable, if and only if its reflex silhouette is empty and its shadow curves are non-crossing.
- Given a polyhedron with n vertices, it can be tested in $O(n \log n)$ time whether the polyhedron is vertically monotone and, therefore, moldable.
- All the possible directions to eject the two mold pieces can be computed in $O(n^4)$ or $O((n^2 + m) \log n)$ time by traversing all the vertices, edges and faces in an arrangement $A(L)$ of great circles and arcs of great circles that denote the critical events in a unit sphere S^2. Here m is the combinatorial complexity of $A(L)$. This algorithm is optimal in the worst case.

Building on this, Elber et al. [Elbe05] have developed an algorithm to solve the problem of determining parting directions for general free-form shapes, represented by NURBS surfaces. Not only have they given an algorithm to find a separating direction, they have also found a valid parting line (i.e. the contact curves of the two mold pieces). Just like Ahn et al. they have also reduced this problem essentially to one pertaining to silhouettes and visibility. A relatively straightforward way to obtain a valid 2-piece mold design is to enumerate all the possible topologies of its silhouette under all orthographic viewing directions, and then to see whether any one of them has exactly $g + 1$ loops, each of which is non-singular. Here g is the genus of the compact surface under consideration. Since this is precisely the information that is contained in vision events and aspect graphs, the next important issue to how to compute them for NURBS surfaces. Although techniques to compute them for different kinds of surfaces like polyhedral, quadric, surfaces of revolution and general implicit surfaces exist in the available literature, Elber et al. have presented the methodology to determine them in case of NURBS for the first time.

They have given a four-step approach to solve this problem. Since vision events occur only when the viewing direction crosses the generating cones of the three developable ruled surfaces, namely axis cylinder developables, flecnodal scrolls and limiting bitangent developables, it is necessary to compute the intersection of their generating cones with the Gauss sphere. From the resulting partition of the Gauss sphere, a representative point may be selected and the silhouette corresponding to that viewing direction is evaluated to check whether it free from singularities and has $g + 1$ loops.

Although this method presents a complete solution to the 2-moldability problem for a model bounded by $C^{(3)}$ surfaces such as NURBS, it cannot directly handle piecewise smooth surfaces. Moreover, this method needs to be extended to handle planar facets present in polyhedral models. Furthermore, this method is time-consuming; it may take several minutes to obtain results on practical parts. Hence, extraction of feature curves and detection of silhouette singularities need to be carried out in a more robust and optimum way.

Recently, McMains and Chen [McMa06] have determined moldability and parting directions for polygons with curved (2D spline) edges using the concepts originally presented in [Rapp94]. They have proposed an efficient algorithm to solve the opposite direction moldability problem. They have introduced a new data structure called normal graph to represent the range of normals of the polygon edges along with their connectivity. It has been proved that this normal graph captures the directions of all lines corresponding to feasible parting lines. However, instead of building a full normal graph which would take O(*n* log *n*) time for a polygon bounded by *n* possibly curved edges, they have built a summary structure in O(*n*) time and space, so that all feasible parting directions can be obtained in O(*n*) time.

Although this method can handle both faceted polygons and those with curved edges using identical graph summary structure and computes all feasible parting directions so that eventually the optimum one can be chosen with respect to other manufacturing considerations such as mold flow, shrinkage, gate/runner placement, parting plane geometry etc. practical parts are always 3D geometries. Hence, this method is not very useful in the current form. In fact, the difficulty with 3D geometries is that there is no natural traversal order of the faces of the polyhedron and 3-moldability depends upon face geometry in addition to face orientation and connectivity. Hence, an alternative way may be to use the analysis for 2D input contours defining 3D extrusion, rotation, loft and sweep for incremental moldability analysis of 3D geometries as and when the user designs them.

Such a scheme has been proposed by Chen and McMains [Chen06] for extrusion operations. They have built upon their previous result to find the exact set of undercut-free parting directions in O(*n* log *n*) time, where *n* is the complexity of the 2D generator profile. However, another slightly conservative superset of the exact set can also be built in O(*n*) time. The salient features of this approach is that the set of possible parting directions for a part containing multiple extruded features can be reduced based upon an analysis of each such feature, parts having no undercut-free parting directions can be efficiently identified and the search time for all possible parting directions is reduced in many cases. However, this still does not consider lofted, swept or revolved features and does not take into account other factors such as planarity of parting surfaces and so on in coming up with an optimal parting direction.

Before concluding this section, another new and novel technique suggested by Kharderkar et al. [Khar06] needs to be mentioned. They have presented new programmable graphics hardware accelerated algorithms to test the 2-moldability of parts. Their running times only grow linearly with respect to the number of facets in the CAD model, thereby enabling testing 2-moldability of tessellated CAD model in real time. Two different algorithms have been developed to obtain the set of candidate directions using Gauss maps, accessibility analysis and visibility detection techniques such as back face culling. The efficiency stems from the fact that groups of candidate directions can be identified such that if any one direction in a group is undercut-free all are, or if any one is not free, none are.

The first algorithm is based upon the algorithm presented by Ahn et al. [Ahn02]. It projects all the Gaussian sphere great circles corresponding to combinatorially distinct parting directions on a plane tangent to the sphere to form an arrangement of lines. Then the line arrangement is sub-divided into a quadtree data structure with 32 lines per quadtree node. Each line is then allocated one color bit and randomly points having different colors (corresponding to distinct 2-cells in the arrangement) are retained. However, this does not guarantee that a direction will be tested in the interiors of all the 2-cells. Moreover, it ignores all the 0 and 1-cells, which may sometimes contain the only undercut free parting directions. Furthermore, the speed of frame-buffer read-back is often too slow for practical applications on parts having large number of facets.

The second, more accurate algorithm uses spherical convex hulls on Gauss sphere. Based on some mathematically involved lemma and theorems, the authors have established that while moving over the Gaussian sphere that encompasses all possible parting directions, the only event that alters the 2-moldability of the current direction is when a pair of potentially interacting facets [see page 333 in Khar06 for definition] start or stop occluding (blocking) each other. Thus, 2-moldability will only change when the current direction crosses one of the arcs bounding the accessibility regions of the potentially interacting facets. These arcs divide the Gaussian sphere into connected regions. In other words, it suffices to test 2-moldability at the vertices of the connected regions to check whether any undercut free parting direction exists for the object. To sum up, it may be said that even though further work needs to be done to eliminate some redundant tests and extend this idea to design multi-piece molds, this work provides an excellent background for application of potent computer graphics technique to solve a manufacturing world problem.

2.2 Determination of Parting Line and Parting Surface

Vast amount of work has been in the field of determination of parting lines and parting surfaces of injection molded parts. Pioneering work was carried out by Tan et al. [Tan90]. Given a solid model of a part (winged-edge boundary representation (B-Rep) or constructive solid geometry (CSG)) and the preferred parting direction, their method automatically generates the parting line and parting surfaces, which would allow the resulting mold halves to be configured. This method also detects interlocking surfaces which will require side cores.

According to the authors, for a simply connected part, parting line is one which contains the outermost points surrounding the part when viewed in the parting direction. A simple and efficient algorithm has been presented to determine this through the evaluation of surface normals and dot products with the parting direction. It includes traversing part faces to classify them as tentatively visible and non-visible and then connecting a series of tentative parting edges (shared by tentatively visible and invisible faces) to form the desired parting line. On the other hand, parting surfaces are obtained by projecting the parting line along the parting direction on a 2D plane to obtain the outer and inner loops and then by taking the convex hull of the outer loop edges. The method then fills the sub-region between the convex hull and the parting edges of the outer loop by inserting some additional planar faces.

However, in order to deal with protrusions, artifacts need to be introduced to form stepped parting surfaces. Moreover, a large number of approximated facets are required to ensure reasonable accuracy for an object with curved surfaces. Theoretically, this method can be extended to curved surfaces by evaluating silhouette curves instead of edge counting. The curved parting lines may need to be split into piece-wise segments and each of these segments will form a parting surface using the approach mentioned above.

Ravi and Srinivasan [Ravi90] first presented a novel approach to enable computer aided design of parting surfaces using a number of design criteria, namely projected area, flatness, draw, draft, undercut features, dimensional stability, flash, machined surfaces and feeders. The correlation between parting surface and different design criteria can be listed as follows:

- Cross-sectional area should gradually decrease from the parting surface to points farthest from the parting surface.
- A flat parting surface is preferred over an irregular one from dimensional stability, sealing off, flash and cost of tooling and mold-making aspects.
- Draw should be as less as possible to minimize metallurgical problems such as grain flow in forgings; small draw also reduces cycle time and increases productivity.
- Number of surfaces requiring drafting should be minimized.
- Undercuts should be avoided wherever possible.
- Parting surface should be so designed that any two points between which higher dimensional tolerance is required occur on the same side of it.
- Parting surface should produce the least number of intersections with faces where occurrence of flash has to be avoided.
- Critical surfaces of a molded part requiring machining should be preferably located as the bottom or vertical walls of the mold.
- Parting surface should be so chosen that the hot spots (regions of mass concentration) are at the top of the molding.

Although the authors have developed a simple optimization framework to design the best possible parting surface based on weights pre-assigned to all the decision criteria, a separate expert system needs to be built to perform this systematically. Moreover, this method is not capable of handling stepped, profiled or complex parting surfaces.

Ganter and Skoglund [Gant93] pointed out that the addition of side cores and core prints to an existing object will affect the placement of parting surface. In other words, side core and core print information must be available concurrently to the process that develops the parting surface. They have presented a technique to allow automatic extraction of three types of side core features, namely, voids, single and multi-surface holes and boundary perturbations. A combination of solid modeling (B-Rep) and graph structures has been used in this technique. Appropriate local features are identified and extracted from the original object and are grouped into one or more new side core objects. Core prints are later added after augmenting the original part geometry by combining the extracted core geometry and its convex hull. Although the authors have dealt with the importance of side cores in parting surface design in details here, it does not take account the other factors.

Weinstein and Manoochehri [Wein97] have presented a methodology to obtain an optimum parting line design for injection molded or die-cast parts based on two manufacturability considerations, namely, parting line complexity, draw depth, number of undercuts, number of unique side cores and mold complexity. In order to quantify these factors two design variables have been used: draw direction range and parting line location. These design variables can be extracted from the part geometry by dividing it into concave and convex regions since they determine the allowable draw range and parting line location for the part respectively. A non-linear programming scheme is then used to generate the optimal mold configuration. In other words, this work is the parting line analogue of the method originally proposed in [Ravi90].

However, any review of the available literature will be incomplete without mentioning the method suggested by Wong et al. [Wong98] to form parting line and parting surface by slicing a CAD model. The salient feature of this work is that it can handle free-form surfaces in addition to planar and curved surfaces. It first selects a set of possible draw directions (typically along the principal axes) and forms tentative parting lines based on the selected draw directions. Then the best parting line is chosen among the plausible candidates based on certain criteria. A slicing algorithm is used to locate the parting line for the model along the possible draw direction. Then an undercut detection algorithm is used locate all the undercuts along the selected draw directions. Finally, the flatness of the parting line, draw distance, projected area, undercut volume, undercut length and relative positions of external undercuts to the mold are used to choose the best parting line. Although this method is computationally efficient, works for all kinds of surfaces and considers all the factors while selecting the best parting line, it does not search all the draw directions exhaustively and hence cannot ensure global optimality.

2.3 Undercut Feature Recognition

Many people around the globe have worked on undercut feature recognition problem using concepts similar to those used for computing parting directions. Yin et al. [Yin01] have tried to recognize undercut features for near net-shape manufactured parts by using local freedom of sub-assemblies in directed blocking graphs (DBGs). They represented contact between two faces hindering accessibility as point constraints set at vertices of convex hull of contacting surface area. Thus, each constraint *C* cuts unit sphere S^2 along a great circle and set of such great circles define an arrangement of cells (namely vertices, edges and faces) on S^2. This unit sphere is obtained from the freedom cone (FC), which is nothing but the set of feasible translation directions in 3-D. This is basically an improvement or modification on VMaps that uses computationally better schemes like J_0 functions and central projections to reduce time-complexity of algorithm from $O(n^6)$ to $O(n^3)$.

Another way of recognizing and especially categorizing undercuts was put forward by Fu et al. [Fu99b]. They classified undercuts further into outside and inside on top of external and internal types. They defined an undercut feature direction as one whereby the local tool can be withdrawn most effectively to avoid warpage of the molding. Although for internal undercuts they have used V-maps only, for outside external and outside internal features, they have considered individual cases in great details and come up with specific formulae based upon normals to the *adjacent surfaces* (Please see Table 1 in [Fu99b] for more details). However, no clear justification is given for choosing a particular direction in each of the individual cases. They have also suggested generating optimal parting direction by maximizing the undercut feature volumes in each undercut feature group in another closely related work [Fu99a].

A hybrid method for recognizing undercut features from moulded parts with planar, quadric and free-form surfaces has been proposed by Ye et al. [Ye01]. They have used a combination of graph-based and hint-based approaches. Undercut features are defined using extended attributed face-edge graphs (EAFEG) and are recognized by searching cut-sets of subgraphs instead of commonly used graph matching techniques. Face properties and parting lines are used as hints to guide the search of cut-sets. Different case studies show that the proposed method can recognize various undercut features successfully and efficiently by avoiding time consuming subgraph isomorphism. However, it is not able to determine the feature parameters, namely, volume and release direction; hence side actions for molding such features cannot be designed directly.

As already discussed in Section 2.1, Kharderkar et al. [Khar06] have presented programmable graphics hardware based algorithms to test the 2-moldability of parts. In addition to testing moldability, these algorithms also identify and graphically display undercuts so that parts can be redesigned immediately. The basic idea behind these algorithms is very simple: if the part is illuminated by two light sources located at infinity along the main mold opening directions, then undercut facets are those which lie in shadow. This is done in real time (of the order of milliseconds) using the depth texture (shadow texture) capability of existing graphics hardware. Thus, this novel approach opens up exciting avenues in detection of undercut features.

Priyadarshi and Gupta [Priy06] have adopted a similar shadow mapping approach to identify all the different mold-piece regions (core, cavity, *both* meaning can be either molded by core or cavity, and undercut) on parts. They have handled near-vertical facets robustly by slightly perturbing the vertices on those facets and by using visibility sampling. They have also managed to overcome the problem of self-shadowing by adapting second depth techniques suitably. The algorithms are very efficient; the time-complexity solely depends on the time to render the parts. To sum up, it may be said that the last two methods using programmable Graphics Processor Units (GPUs) outperform the traditional software based methods both in terms of granularity (resolution or accuracy) and speed.

2.4 Design of Side Cores

Quite surprisingly, automatic design of side actions has not received a lot of attention to date. However, Shin and Lee [Shin93] developed a procedure by which *side* *cores* (specific types of side actions) and the corresponding core and cavity plates are generated by identifying the mold faces that prevent interference free motion of the part after solidification. These interference faces (both primary and secondary) are derived directly from the core and cavity plates without any need for considering the part by splitting faces along silhouette curves and applying polygonal overlap tests. Since the algorithm uses Euler operations like kill edge and face, kill edge and vertex etc. directly rather than Boolean operations, they are inherently efficient. However, this method cannot represent all the normal vectors (required for later computations) of a free-form surface by selective sampling. Moreover, it is not fully automatic. Many faces need to be selected interactively and the retraction or sliding direction has to be specified by the designer. In short, although this method attempts to solve this challenging problem in mold design for the first time, it does not offer a comprehensive solution that can be readily used in the industry.

In the more recent past, Ye et al. [Ye04] have extended their idea of using EAFEG to recognize undercut features (as mentioned in Section 2.4) to generate side cores for each undercut region by using certain Boolean operations. They have also used VMaps to determine release directions. Then a bounding box is created to enclose the blockage portion of the undercut feature; it is trimmed with all the faces of the feature and then swept along the release direction to get a linkage portion. This linkage portion is finally unionized with the blockage portion to form the actual side core. Although this approach fails to consider all the possible release (retraction directions) directions, a similar sweep operation will be used in the current work to generate actual solid shapes.

2.5 Design of Multi-piece and Multi-Stage Molds

As discussed briefly in sub-Section 1.2.1, multi-piece molds offer an alternative to two-piece molds involving side actions. Broadly speaking, they can be of two types- permanent and sacrificial. Some of the representative work in both types of multi-piece molding is briefly discussed below.

Chen et al. [Chen02, Chen03] first provided an excellent groundwork for permanent multi-piece mold design that allowed three-dimensional mold decomposition. They subdivided the process into two subsequent processes: mold configuration design process and mold construction process. In the first step, object boundary is sub-divided into smaller regions (mold-piece regions) that will be created by different mold pieces. Parting direction is found for each mold piece region by solving a linear optimization problem. However, if it does not have a parting direction, then it is split into concave regions and convex faces. Then in the second step, an approach based on reverse glue operation is used. Selection of different glue faces and planar parting surfaces produce different mold pieces recursively.

However, their work suffers from two major limitations. First, since a local approach has been followed to find the parting direction of a region, disassembly of mold pieces is not guaranteed. A separate interference test simulation module is required to verify the feasibility of mold design generated by this approach. Second, as a greedy heuristic is used to select the mold-piece regions, it may not always produce the optimum solution.

Priyadarshi and Gupta [Priy04] successfully tackled some of the problems in the previous work by proposing a 4-step approach for automated design of multi-piece permanent molds. The steps are listed as follows:

- Finding candidate parting directions based on an initial set of heuristic directions followed by a more computationally intensive method of obtaining directions from global accessibility cones suggested by Dhaliwal et al. [Dhal03].
- Analyzing accessibility of every facet in the part along at least one direction belonging to the set of candidate parting directions by using separating axis condition. Special attention has been paid to robustly determine the accessibility of near-vertical facets.
- Selecting near-optimal mold-piece regions from the set of candidate mold-piece regions by applying depth-first branch and bound and set covering heuristics.
- Constructing mold pieces by first computing the gross mold, then locating the parting line for a mold-piece region, creating parting surface for the mold-piece region and finally splitting the gross mold along the parting surface recursively.

Although this provides a significant improvement over the previous approaches with respect to various characteristics such as soundness, completeness, efficiency, solution quality and domain of applicability, it has two major limitations. It cannot handle free-form surfaces without prior faceting and is unable to deal with faces that can be molded using split cores.

Now coming to sacrificial multi-piece molds, Dhaliwal et al. [Dhal01] presented a feature-based approach. For those objects which cannot be represented by feature-based models, this approach provides a 3D partitioning scheme to solve this problem in a computationally efficient manner. However, it cannot be used to design molds for arbitrarily complex parts.

Huang et al. [Huan03] describe an algorithm based on accessibility-driven partitioning approach to design multi-piece sacrificial molds. After performing accessibility analysis of the gross mold, it is partitioned using the accessibility information. During each partitioning, a set of mold components is produced. Since each mold component is accessible, it can be produced using milling and drilling operations. However, in both these methods although the mold pieces are manufacturable, they may not be disassemblable.

As briefly mentioned in Section 1.1, multi-material (multi-stage) molding is rapidly becoming popular in industry due to its various benefits. Pioneering work in developing an algorithm for designing multi-stage molds was carried out by Kumar and Gupta [Kuma02]. In order to obtain a feasible molding sequence, this algorithm decomposes the multi-material object into a number of homogenous components that can be added sequentially to produce the desired overall product. Once an object-decomposition scheme has been found, the gross mold for each stage is computed and split into two or more mold-pieces.

Novel features of this algorithm include: (1) it finds multiple partitioning planes to perform partitioning of the mold pieces, (2) it performs object and mold decompositions needed to ensure assembly and disassembly of mold pieces during actual molding operation and (3) it generates complete molding sequence for multi-stage molds. However, it has some limitations as well. First, the contact between homogenous components is assumed to take place through planar faces only. This restricts the type of material interfaces in the multi-material object that can be currently handled by this method. Second, the object decomposition algorithm does not always find a feasible object partitioning scheme because it only decomposes components along the material interfaces.

Certain improvements over the last approach for automated design of rotary-platen multi-shot molds were suggested by Li and Gupta [Li04]. First, this method classifies any given multi-material object into several basic types based on the relationships among different components in the object. Then for every basic type, a molding sequence is obtained depending upon the precedence constraints resulting from both accessibility and disassembly requirements. Then starting from the last mold stage, mold pieces are generated for every mold stage.

The salient features of this work are: (1) constraints for the rotary-platen process are used in generating molding stages, such that the generated mold can be used in an industrial process, (2) significantly more complex curved interfaces can be handled by using the exact component geometry in generation of parting surfaces and (3) disassemblability of mold pieces is always guaranteed. However, there are a few limitations as well. First, the object cannot be made of more than two materials. Second, the mold pieces generated by this algorithm may not have the optimal shapes.

2.6 Die Design and Other Related Work

For the sake of completeness, let us review a representative work in the field of die design. Die design broadly refers to the design of dies to produce desired part geometries in all near-net-shape manufacturing processes such as casting, forging and injection molding. The die must be designed such that it is reusable and can meet the manufacturing requirements strictly. To address these needs, use of virtual prototyping is becoming increasingly popular in the industry.

An approach for virtual prototyping of die design has been proposed by Wuerger and Gadh [Wuer97a, Wuer97b]. It uses concavity features to determine the set of all possible die-open directions for a given part by first determining the convex hull of the part and then by performing Boolean subtraction between the convex hull and the solid model of the original part geometry. VMaps are used to display all the possible die-open directions for a given part. The user can then select a single direction, based upon which the parting line will be developed. Finally, taper and draft need to be added to the part. Although this method may well serve as the foundation for automated die-design process, it is currently unable to split a concavity feature so that it is produced by two die-halves.

2.7 Summary

To summarize it can be said that lot of work has been done in automated design of multi-piece and multi-stage molds. The problem of finding optimum parting direction has also been studied extensively and is very useful in casting applications. However, in case of two-piece SM injection molded plastic parts, most designers develop part designs with a mold opening direction in mind. This is because mold opening direction influences all aspects of part design. Typically, plastic parts (few representative parts are shown in Figure 2.1) are either flat or hollow-box shaped and they need to have relatively thin sections. These shapes have an implied mold opening direction. Moreover, they cannot have vertical walls; and some amount of taper has to be imparted in order to remove them from the mold assembly. Therefore the main tasks in two-piece SM injection mold design automation are (1) determination of the main parting line and parting surface, (2) recognition of undercuts, and (3) design of side actions. Existing methods for the first two tasks provide satisfactory solutions. However, existing methods for side action design only appear to work for certain class of undercuts.

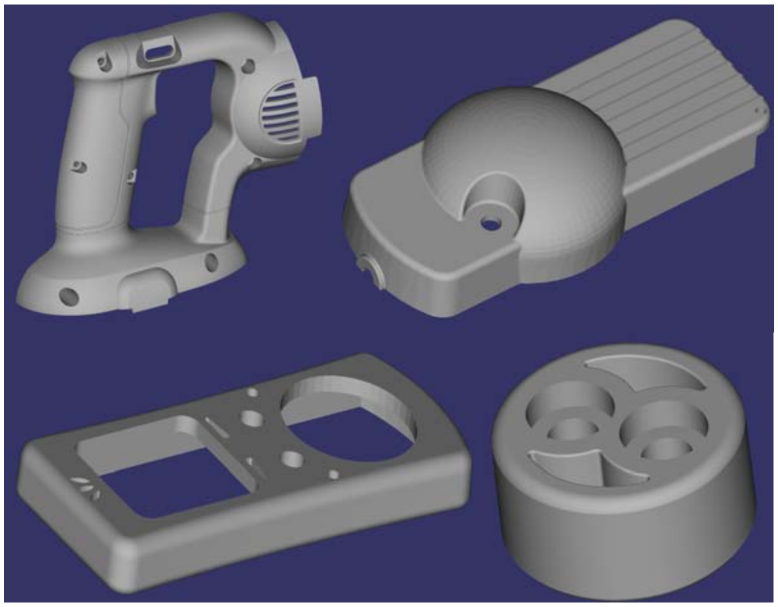


Figure 2.1: Typical industrial parts

Existing methods for side action design do not work satisfactorily (i.e. cannot guarantee an optimal solution from an exhaustive set of feasible solutions) if a complex undercut region (1) needs to be partitioned to generate side actions, or (2) has finite accessibility or (3) is impossible to mold, thereby requiring the part to be redesigned. Figure 2.2 shows three parts for which generating side actions is challenging due to these reasons. In this work, new algorithms have been presented to handle these types of cases. Currently only those types of side actions are considered that are initially retracted from the undercut in a direction perpendicular to the main mold opening direction. In fact, majority of side actions used in industrial parts meet this restriction. Typically, a single side action produces a connected undercut region. Therefore, each connected undercut region is treated independently and appropriate number of side actions is generated for molding it.

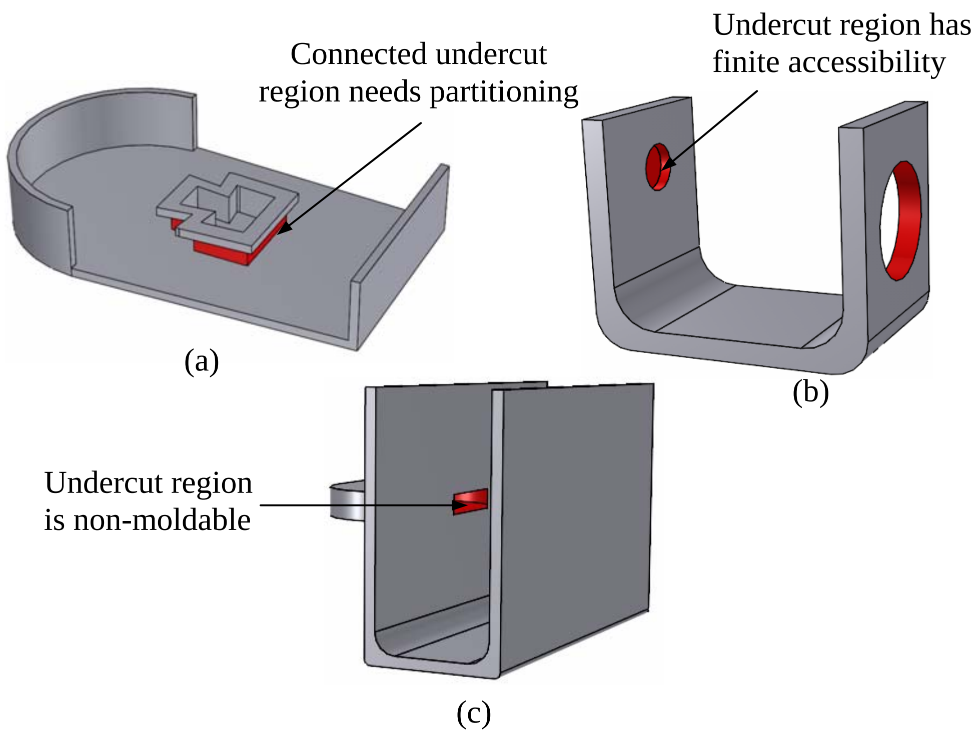

Connected undercut region needs partitioning
Undercut region has finite accessibility
(a)
(b)
Undercut region is non-moldable
(c)

Figure 2.2: Parts that pose challenges for existing side action design algorithms

Chapter 3

PROBLEM FORMULATION AND THEORETICAL FOUNDATIONS

This chapter formulates the problem of generating side actions and establishes the theoretical framework for coming up with efficient algorithms to do the same automatically. Section 3.1 introduces and defines certain basic terminologies which will be used extensively later on. Section 3.2 gives the input and output of the algorithm along with the input restrictions and output requirements. One of the output requirements is to minimize an objective function. Section 3.3 derives this objective (molding cost) function from *ab initio*. Finally Section 3.4 gives a detailed overview of the approach as well as mathematically verifiable lemmas and theorem to characterize it.

3.1 Definitions

Retraction is defined by a point (*x*, *y*) in 2D configuration space. We are interested in two types of retractions namely, core-retractions and cavity-retractions.

Retraction length is defined as the length of the position vector of the 2D point that defines the corresponding retraction.

A given retraction (*x*, *y*) is a *candidate core retraction* for an undercut facet *f*, if sweeping *f* first by a translation vector $\overrightarrow{t_1} = x\hat{i} + y\hat{j}$ and then by another one $\overrightarrow{t_2} = l_a\hat{k}$, (where l_a is a large positive number) does not result in any intersection with the part. Figure 3.1 shows examples of a candidate and a non-candidate core retraction.

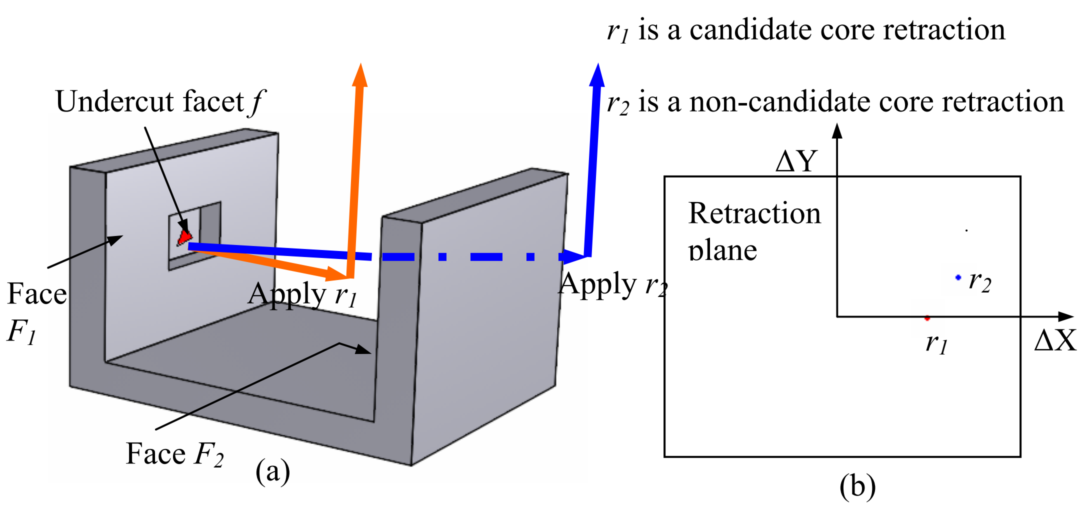


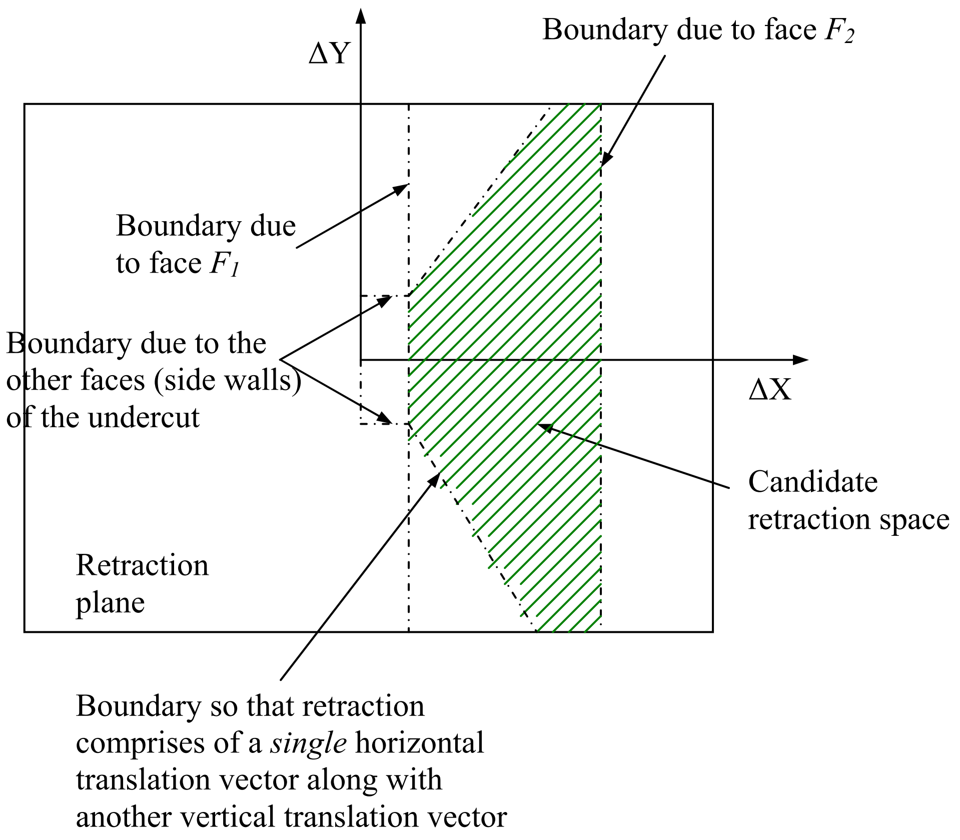
Figure 3.1: Candidate and non-candidate core retractions

A given retraction (x, y) is a *candidate cavity retraction* for an undercut facet f, if sweeping f first by a translation vector $\vec{t}_1 = x\hat{i} + y\hat{j}$ and then by another one $\vec{t}_2 = -l_a\hat{k}$, does not result in any intersection with the part.

Candidate core retraction space (illustrated in Figure 3.2) for an undercut facet f is the set of all candidate core retractions for that particular facet. *Candidate cavity retraction space* is defined in a similar manner.

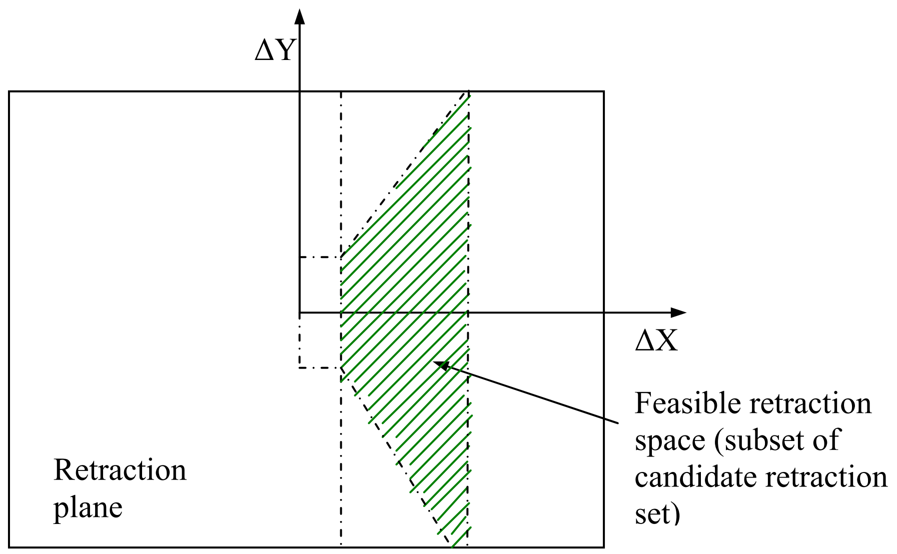
A given retraction (x, y) is a *feasible core retraction* for an undercut facet f, if in addition to being a candidate core retraction it also satisfies the condition that the swept polyhedron SP does not intersect with the part where $SP = \{q + \alpha\vec{t} : q \in f, 0 \le \alpha \le 1\}$ and $\vec{t}$ is the minimum length translation required to reach a point in the mold free space such that this point is accessible along $+z$. *Feasible cavity retraction* is defined analogously by imposing this additional constraint on candidate cavity retraction.

Feasible core retraction space (illustrated in Figure 3.3) for an undercut facet f is the set of all feasible core retractions for that particular facet. Similarly, *feasible cavity* *retraction space* is defined as the set of all feasible cavity retractions for an undercut facet *f*.



ΔY
Boundary due to face F_2
Boundary due to face F_1
Boundary due to the other faces (side walls) of the undercut
ΔX
Candidate retraction space
Retraction plane
Boundary so that retraction comprises of a *single* horizontal translation vector along with another vertical translation vector

Figure 3.2: Candidate core retraction space



ΔY
ΔX
Feasible retraction space (subset of candidate retraction set)
Retraction plane

Figure 3.3: Feasible core retraction space

3.2 Problem Statement

Side action set generation problem: Given a polyhedral object, its *mold enclosure* (bounding box enclosing the part), main mold opening directions and undercut facets, determine the side action(s). Side actions require two translational motions for complete disengagement from the part. The first one is assumed to lie in a plane orthogonal to the main mold opening directions and the second one coincides with the either of the two main mold opening directions. Furthermore, it also assumed that all the side actions can be actuated simultaneously, independent of one another.

Input:

- A faceted (triangulated) solid geometric model of a polyhedral connected part *P* oriented such that the main mold opening directions are along +*z* and –*z* respectively.
- Mold enclosure.
- *n* facets marked or labeled as undercuts (we use technique described in [Priy06]).

Output:

- A set of side actions, such that each set (SAS) is defined by a 4-tuple $\{s, F, (x, y), T\}$. The first entity *s* represents a solid body corresponding to the side action. The second entity *F* represents the set of undercut facets that will be associated with this side action. The third entity (x, y) is a retraction and the fourth entity (an integer) describes whether this is a core or a cavity retraction (+1 for core and -1 for cavity). Sample output for a given part is shown in Figure 3.4.

Input restrictions:

- Faceting is done on the part *P* with a sufficient degree of resolution so that each undercut facet does not have any point on it, which is accessible along either +*z* or –*z*.

Output requirements:

- An undercut facet *f* must be included in one and only one side action facet set *F*.
- All the facets marked as undercuts must be associated with one of the side action facet sets *F*.
- (*x*, *y*) is a feasible core or cavity retraction for all facets in *F* based on the type defined in *T*.
- Any two side action solids will not intersect.
- The side actions generated will minimize the following objective (molding cost) function:

$$C = \gamma \sum_{i=1}^{N} \left| x_i \hat{i} + y_i \hat{j} \right|^{k} + NC' + \chi \sum_{i=1}^{N} \left| x_i \hat{i} + y_i \hat{j} \right| \qquad (3.1)$$

where N is the cardinality of the output set, (x_i, y_i) is the retraction in the ith element of the set, and γ, k, C', χ are molding parameters. Derivation of this equation as well as the physical significance of molding parameters is explained in Section 3.3. It is important to note that depending on the geographical region and the nature of the molding operation, values of these parameters will be significantly different for the same part. In other words, these parameters might have altogether different values for production-run molds and prototyping molds even for the same part.

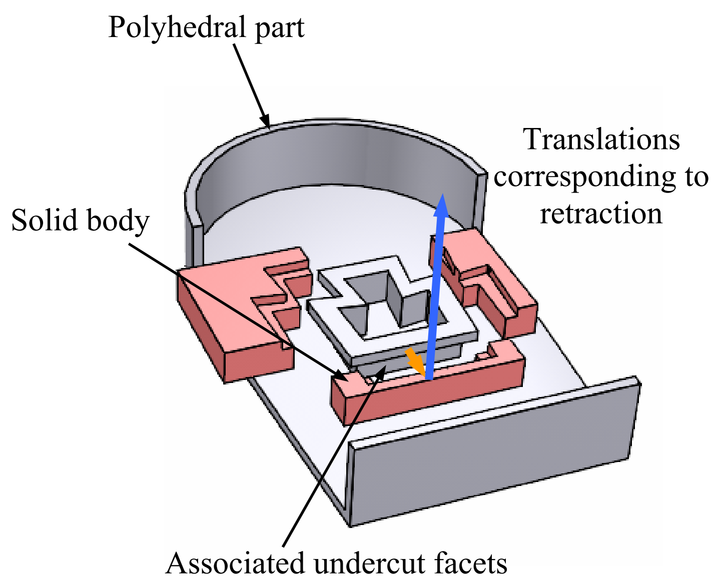

Polyhedral part
Translations corresponding to retraction
Solid body
Associated undercut facets

Figure 3.4: Sample side action set (output)

3.3 Derivation of Cost Function

Several industries as well as academic researchers [Boot02, Poli01, Faga00, Bryc96, Rosa00] have propounded different models to estimate the total cost in plastic injection molding. The current work starts from a very simple model [Poli01] and develops a unique cost function that serves as the objective function for the problem of grouping the undercut facets into optimal number of groups. Since total cost = Molding cost + Operation cost + Material cost, symbolically,

$$C = C_{MO} + C_{OP} + C_{MA} \tag{3.2}$$

where C, C_{MO}, C_{OP} and C_{MA} refer to the total cost, molding cost, operation cost and material cost respectively in dollars.

Assuming that the injection molding shop floor is going to produce Q parts, each of these cost components can be broken down into sub-components which relate the overall cost function to certain input parameters and part properties (material and geometry). As molding involves manufacturing and assembling the two main mold pieces as well as the side cores, C_{MO} can be sub-divided as follows:

$$C_{MO} = C_{MP} + C_{SC} \tag{3.3}$$

where C_{MP} and C_{SC} represent the individual costs of the main mold pieces and the side cores respectively. Since the mold pieces need to be manufactured and assembled only once, these costs are independent of the number of parts to be produced. In our model, C_{MP} is taken as a constant and will be inputted by the mold manufacturer as the total cost of manufacturing (milling, grinding, EDM etc.) core and cavity. However, the side core cost is a variable one and has three components. Thus,

$$C_{SC} = C_{mac} + C_{act} + C_{ass} \tag{3.4}$$

where C_{mac}, C_{act} and C_{ass} denote the costs due to machining, actuating and assembling the side cores respectively.

Actuation and assembly costs can be grouped together for the sake of simplicity and modeled as being proportional to the number of side cores being used. However, core-side side cores are typically actuated by solenoid based mechanisms, whereas cavity-side side cores are actuated by slider mechanisms. Thus, this cost varies depending upon the nature of side cores. The cost associated with core-side side cores is naturally much higher than that corresponding to cavity-side side cores. Hence, this can be mathematically formulated as follows:

$$C_{act} + C_{ass} = N_{Co} C_{Co} + N_{Ca} C_{Ca} \tag{3.5}$$

where N_{Co} and N_{Ca} are the number of core-side and cavity-side side cores respectively. C_{Co} and C_{Ca} are proportionality constants (in dollars) to relate the combined actuation and assembly cost with the number of core and cavity-side side cores respectively.

Now coming to the machining cost C_{mac}, it comprises three major components- namely cost due to the size or volume of the machined part, cost due to machining the cross-section and cost due to the complexity of the machined profile. Hence, C_{mac} can be mathematically represented as:

$$C_{mac} = \alpha \sum_{i=1}^{N} V_i + \beta \sum_{i=1}^{N} A_i^{CS} + \gamma \sum_{i=1}^{N} C_i \tag{3.6}$$

where N = total number of side cores

V_i = volume of the ith side core

$A^{CS}{}_i$ = cross-sectional area of the ith side core

C_i = complexity of the profile of the ith side core

α, β and γ are proportionality constants

Now going back to the second component on the right hand side of equation (1), it can be decomposed broadly into two parts- cost incurred due to time taken in injecting, cooling and resetting the mold and the extra cost arising due to retraction of the side cores. Pulling the side cores increases the mold cycle time and thereby raises the operation cost. Hence, C_{OP} may be rewritten as:

$$C_{OP} = \dot{c}_{OP}\, Q \left[\left(t_{inj} + t_{cool} + t_{reset} \right) + \frac{\sum_{i=1}^{N} \left| \vec{t}_{i1} \right|}{v} \right] \tag{3.7}$$

where $\dot{c}_{OP}$ = operation cost rate (\$/hr-piece)

t_{inj} = time required to inject the molten plastic in the cavity

t_{cool} = time to cool the injected molten plastic

t_{reset} = time to reset the solidified plastic inside the cavity

v = speed of retracting the side cores

$\boldsymbol{t}_{i1}$ = horizontal pull (or translation vector) for the ith side core

Again the individual operating times can be expressed in terms of more fundamental process parameters as follows [Bo02]:

$$t_{inj} = \frac{V_{shot}}{q_{avg}} = \frac{2P_{inj}V_{shot}}{\pi_{inj}} \tag{3.8a}$$

$$t_{cool} = \frac{h_{\max}}{\pi^2\alpha}\ln\left(\frac{4\left(T_{inj}-T_{mold}\right)}{\pi\left(T_{ej}-T_{mold}\right)}\right) \tag{3.8b}$$

$$t_{reset} = 1+\frac{7}{4}t_{dry}\sqrt{\frac{2d+5}{L_{stroke}}} \tag{3.8c}$$

where V_{shot} = volume of molten plastic injected in a single shot

q_{avg} = average molten plastic flow rate

P_{inj} = injection pressure

π_{inj} = power output of the injection unit

h_{max} = maximum part wall thickness

α = coefficient of thermal diffusivity of the plastic

T_{inj} = recommended plastic injection temperature

T_{mold} = recommended mold wall temperature

T_{ej} = temperature at which the part may be safely ejected

T_{dry} = time needed for machine to complete a dry run

d = part depth along mold opening direction

L_{stroke} = maximum stroke of clamp unit

Since all these terms are fixed for a particular choice of injection molding process and a given part, the three time components can be combined together to simplify equation (3.7) as follows:

$$C_{OP} = \dot{c}_{OP} Q \left[t_{tot} + \frac{\sum_{i=1}^{N} \left| \overrightarrow{t}_{i1} \right|}{v} \right] \tag{3.9}$$

Finally, the last term in the RHS of equation (1) can be rewritten as the product of material cost per piece (c_{MA}) and the number of pieces to be manufactured as follows:

$$C_{MA} = c_{MA} Q \tag{3.10}$$

Combing equations (1), (2), (3), (4), (5), (8) and (9) we have,

$$C = C_{MP} + \alpha \sum_{i=1}^{N} V_i + \beta \sum_{i=1}^{N} A_i^{CS} + \gamma \sum_{i=1}^{N} C_i + N_{Co} C_{Co} + N_{Ca} C_{Ca} + \dot{c}_{OP} Q (t_{tot} + v \sum_{i=1}^{N} \left| \overrightarrow{t}_{i1} \right|) + c_{MA} Q \tag{3.11}$$

Grouping terms, the final form of the cost function may be obtained as follows:

$$C = \alpha \sum_{i=1}^{N} V_i + \beta \sum_{i=1}^{N} A_i^{CS} + \gamma \sum_{i=1}^{N} C_i + N_{Co} C_{Co} + N_{Ca} C_{Ca} + \chi \sum_{i=1}^{N} \left| \overrightarrow{t}_{i1} \right| + \lambda \tag{3.12}$$

where the constants $$\chi = \dot{c}_{OP} Q v \tag{3.13a}$$

and $$\lambda = C_{MP} + \dot{c}_{OP} Q t_{tot} + c_{MA} Q \tag{3.13b}$$

Since, this molding cost function will be used to compare between two solutions at a time, the term κ can be safely dropped. Moreover, comparisons will be made only between two feasible core or two feasible cavity retractions. Hence both C_{Co} and C_{Ca} cannot be present in C simultaneously and a common symbol $C^{/}$ can be used for them. Finally, the complexity of every side action can be related to the corresponding retraction length. It is assumed that $Cp_i = \left| \overrightarrow{t}_i \right|^k$ where the value of the exponent k will be suitably based on experimental data for that particular molding operation. Therefore, the cost function reduces to:

$$C = \gamma \sum_{i=1}^{N} \left| \overrightarrow{t}_i \right|^k + NC^{/} + \chi \sum_{i=1}^{N} \left| \overrightarrow{t}_{i1} \right| \tag{3.14}$$

which is identical to Equation (3.1).

3.4 Overview of Approach

Before going into the technical details, it is important to understand why the concept of a candidate and a feasible retraction space has been introduced. Candidate retraction space gives the set of retractions such that the corresponding undercut facet can be retracted, first horizontally and then vertically, without intersecting with the part. However, a solid body (side action) of finite dimension is also going to be attached to every undercut facet at the time of retraction. This solid body will extend up to the core or cavity. Thus, it also needs to be ensured that this solid shape never intersects with the part during the process of retraction and that is taken care of in case of feasible retraction spaces. Figure 3.5 shows that if this solid shape *s* is not considered, then *r* is a feasible retraction for the undercut facet *f*. However, as soon as the effect of the attached side action *s* is incorporated, *r* becomes infeasible as it collides with face F_2 at the time of retraction. This clearly establishes the difference between candidate retraction and feasible retraction and their corresponding spaces.

Now as can be inferred from their names, for each undercut facet only its candidate retraction space will be computed (see Chapter 4 for details on this step) and not its feasible one. This space can be represented as a set of one or more disjoint polygons in 2D translation space (ΔXY plane). This translation space can be bounded on all the four sides by the mold enclosure and is termed as the *retraction plane*. Eventually, one gets a set of candidate retraction spaces on the retraction plane. A simple arrangement of two candidate retraction spaces is shown in Figure 3.6.

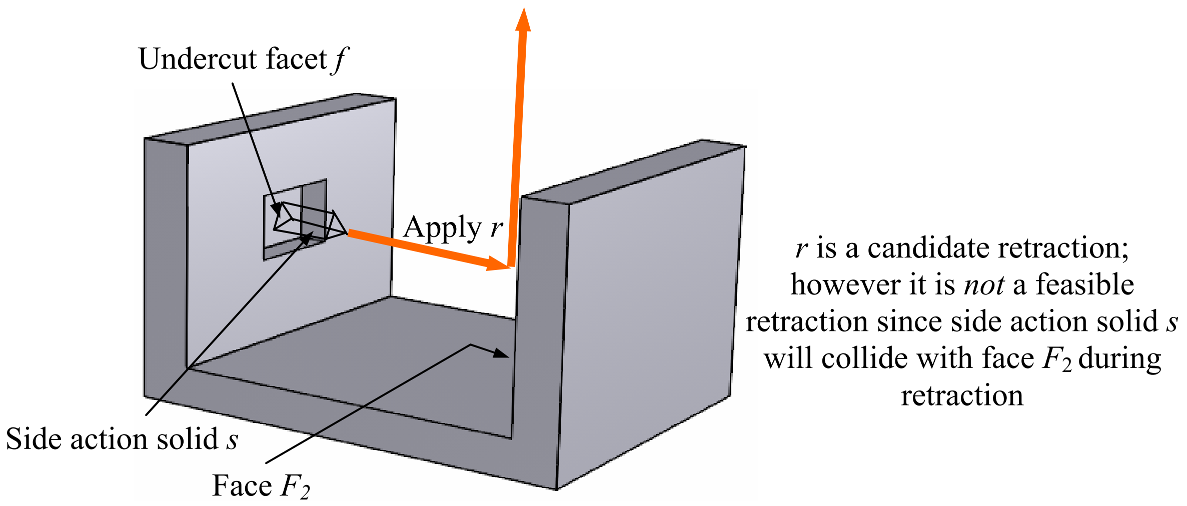

Undercut facet f
Apply r
r is a candidate retraction; however it is *not* a feasible retraction since side action solid s will collide with face F_2 during retraction
Side action solid s
Face F_2

Figure 3.5: Difference between candidate and feasible retractions

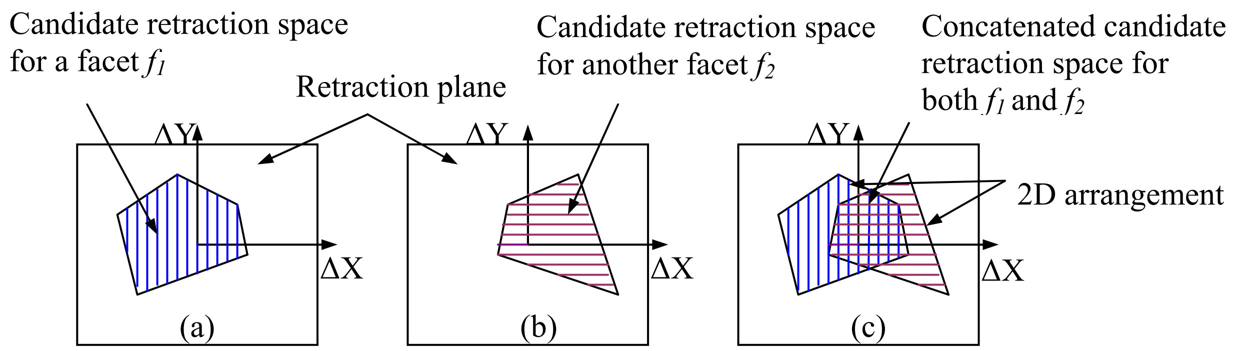

Candidate retraction space for a facet f_1
Retraction plane
Candidate retraction space for another facet f_2
Concatenated candidate retraction space for both f_1 and f_2
2D arrangement
ΔY
ΔX
(a)
(b)
(c)

Figure 3.6: Intersection of two candidate retraction spaces

However, the point to be noted here is that feasible retraction space is a sub-set of the candidate retraction space (as proved in Lemma 1 below). It also satisfies some other unique property (again stated formally in Lemma 1) which makes it possible to compute an exhaustive and discrete set of candidate retractions (to be used for obtaining optimal solution) from the candidate set only. In other words, there is no need to generate the feasible retraction spaces. This clever strategy allows one to bypass many challenges of computing the feasible retraction spaces without compromising on the correctness and feasibility of the solution.

Lemma 1: Feasible core (cavity) retraction space *FS* for an undercut facet *f* is a subset of the candidate core (cavity) retraction space *CS*. Moreover, every element (can be 0-face, 1-face or 2-face) $p_1 \in FS$ will include a portion of the boundary of a unique, corresponding element $p_2 \in CS$.

Proof: This lemma will be proved only for feasible core retraction space. The proof for feasible cavity retraction space follows similarly. From the basic definitions, one can say that feasible core retraction space *FS* for a given undercut facet *f* needs to satisfy another constraint in addition to those satisfied by the candidate core retraction space *CS* for the same facet *f*. Hence, *FS* must be a subset of *CS*. The two sets will be identical if this additional constraint is satisfied for every retraction present in *CS*. For some elements of *CS* the corresponding element in *FS* will be a null element if none of the retractions present in that element satisfies the additional constraint. This proves the first part of the lemma.

As shown in the previous paragraph, p_1 will be a subset of p_2. However, p_1 cannot lie entirely in the interior of p_2. It must include at least some points on the boundary of p_2 because if an interior point of p_2 satisfies the extra constraint, then there will exist a corresponding boundary point which will also satisfy the extra constraint. This corresponding point is the intersection of the line joining the origin to the interior point with the boundary of p_2. Now it will be shown that this boundary point will also satisfy the extra constraint.

This follows from the fact that the retraction length associated with any interior point of p_2 is greater than the length associated with the corresponding boundary point. Hence, if the swept polyhedron *SP* for the undercut facet *f* does not intersect the part while undergoing a retraction by an amount (length) specified by the interior point of p_2, then it can never intersect with the part while undergoing a smaller length retraction along the same direction as given by the corresponding boundary point of p_2. So the corresponding boundary point must also be feasible. This proves the second part of the lemma.

Now, the objective function for this problem calls for minimizing the number of side actions as well as the retraction lengths. Therefore, usually a compromise needs to be worked out by identifying a suitable set of feasible retractions. The set of candidate retraction spaces can be partitioned into a set of cells and let *A* to be the spatial (planar) arrangement defined by them (shown in Figure 3.7). As usual, the arrangement consists of 0-faces (vertices), 1-faces (edges) and 2-faces (polygonal cells). This arrangement *A* is computed by intersecting and splitting the candidate retraction spaces for all the undercut facets. Hence, for each cell *a* in *A*, the set of undercut facets for which this cell will be a part of its corresponding candidate retraction space is known. This set of facets is termed as the retractable facets for the cell *a*. In other words, a subset of all retractions defined by points located within *a* are feasible retractions for *all* the retractable facets. Since by selecting such a retraction, one will be able to deal with a large number of retractable facets at one go, one can then perform a search over all cells and find the optimal combination of retractions. Based on Lemma 1 and the discussion in this paragraph, one can state Observation 1.

Observation 1: For every cell $a \in A$, there exists a corresponding cell a' (may be null (ϕ)) belonging to the feasible retraction spaces, such that $a' \subseteq a$ and $b(a') \cap b(a) \neq \phi$, if $a' \neq \phi$, where b (a') and b (a) are the boundaries of a' and a respectively.

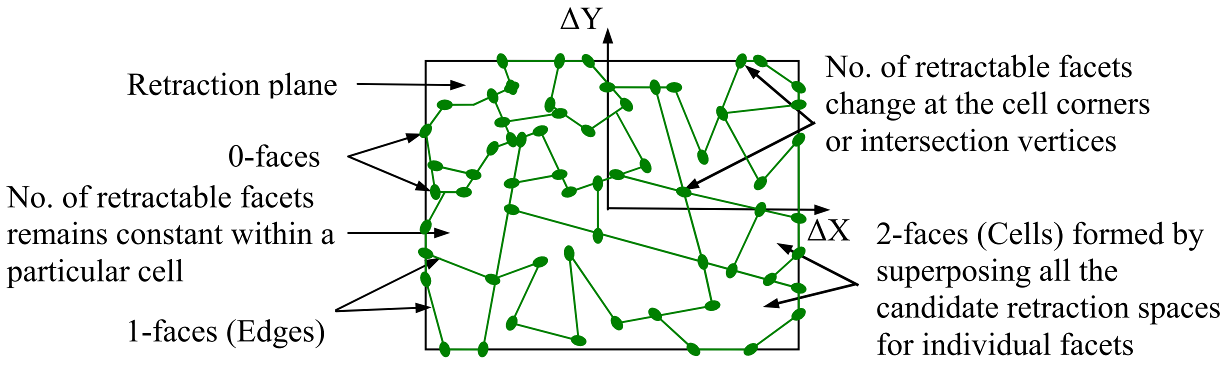

ΔY
Retraction plane
0-faces
No. of retractable facets remains constant within a particular cell
1-faces (Edges)
No. of retractable facets change at the cell corners or intersection vertices
ΔX
2-faces (Cells) formed by superposing all the candidate retraction spaces for individual facets

Figure 3.7: Cellular planar arrangement

Although, the described methodology is followed in spirit, an attempt to explicitly generate A leads to many implementation challenges. Maintaining a doubly-connected edge list data structure for A is a rather difficult and time-consuming task. As de Berg et al. [deBe00] point out, it takes $O(l^2)$ time, where l is the total number of lines in the plane. On the other hand, all pair-wise edge intersections can be reported in $O(l \log l + I \log l)$ time, where I is the number of intersection points. This difference in time-complexities will prove significant since a large number of small cells are expected to be present in practice. Moreover, there is the problem of implementation robustness. If two very narrow or thin cells intersect to create a very small cell, all the neighboring vertices will lie so close to one another that practically it becomes extremely hard to distinguish among them using typical floating-point arithmetic. Such numerical inaccuracies occur frequently in geometric computations, leading to completely erroneous results. This has been illustrated in Figure 3.8. As reported by Kharderkar et al. [Khar05], although such problems can be avoided by using the MP_Float (multi-precision float) number type with Cartesian kernel provided by CGAL [2004], speed becomes very slow due to large overhead and often the application may run out of memory.

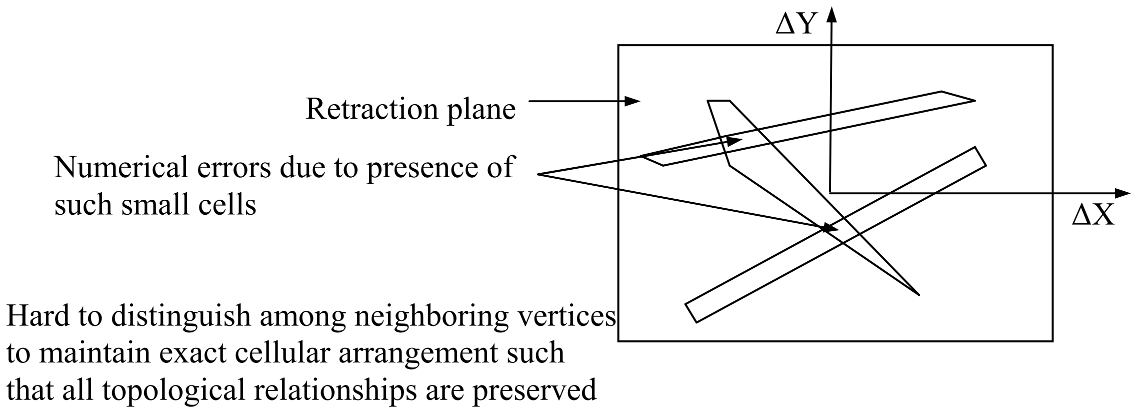
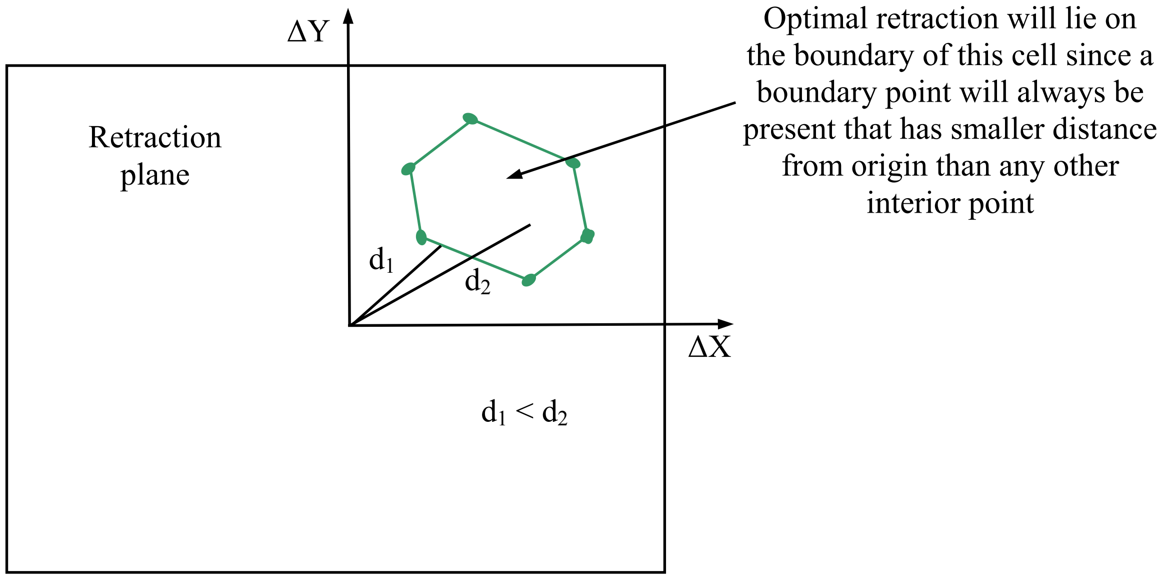

ΔY
Retraction plane
Numerical errors due to presence of such small cells
ΔX
Hard to distinguish among neighboring vertices to maintain exact cellular arrangement such that all topological relationships are preserved

Figure 3.8: Implementation challenges due to robustness problems

Hence, the planar arrangement *A* is not computed explicitly. Instead, focus is given on finding a discrete set of promising retractions and performing search over them. In the following chapters methodology to do the same has been explained. But before proceeding any further, it is useful to establish some important foundations and properties.

Lemma 2: Let *r* be a retraction used in the optimal solution and *F* be the set of facets associated with *r*. Then *r* will lie on the boundary of a cell in *A*.

Proof: Since the objective (molding cost) function is being minimized, if retraction *r* belongs to the optimal solution, then it must be a feasible retraction and the retraction length must have the least possible value for the associated facet set *F*. Since *F* corresponds to a particular cell in *A*, *r* must lie on the boundary of that cell. As shown in Figure 3.9, this is due to two reasons. First, the point having minimum distance from the origin (i.e. retraction length) for any polygonal cell will be a boundary point. Second, the corresponding cell in feasible retraction space will share a portion of the boundary (see Observation 1). Hence the assertion of lemma directly follows from these observations.



Optimal retraction will lie on the boundary of this cell since a boundary point will always be present that has smaller distance from origin than any other interior point
ΔY
Retraction plane
d_1
d_2
ΔX
$d_1 < d_2$

Figure 3.9: Illustration of Lemma 2

This lemma clearly points out that only the boundary of the cells in *A* needs to be considered in order to obtain the least possible retraction lengths for all potential facet sets that may be present in the optimum solution. However, it also needs to be ensured that by considering the cell boundaries only, one does not fail to include maximum possible number of facets in every facet set for all promising retractions that may appear in the optimum solution. Now let us consider a set *R* that consists of two types of elements- the original corner vertices of all the candidate retraction space polygons and the intersection points obtained by pair-wise intersection of all edges in the feasible retraction space polygons. Thus, this set *R* includes all the vertices of the arrangement *A*.

Lemma 3: Let *F* be a facet group in the optimal solution. Then there will exist a retraction *r* in *R* such that *r* is a feasible retraction for *all* facets in *F*.

Proof: The set of retractable facets corresponding to a particular cell does not change within its interior. It definitely changes at the 0-faces, i.e. at the original corner vertices of the free space polygons or at the intersection points. The status of the edges is rather hard to determine. While the boundary edges (edges belonging to a single cell only) have the same set of facets as the respective cell interiors, edges that are common to multiple cells have an ambiguous status. However, this is immaterial here since any change in the retractable facet set along an edge is already accounted by the two vertices forming the edge.

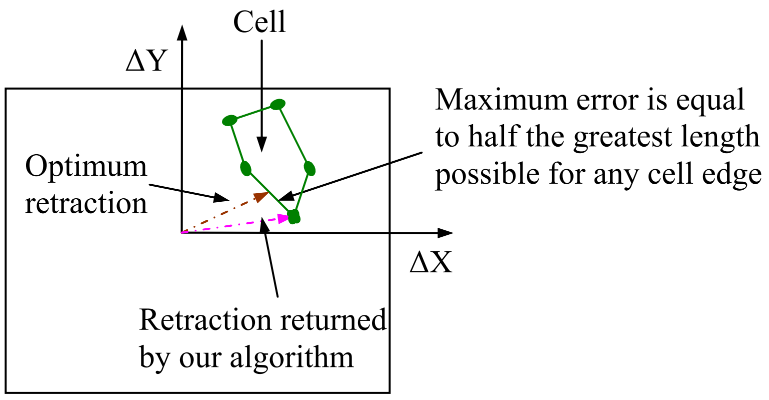
Since *F* is a facet group present in the optimal solution, if retraction *r* is a feasible retraction for all facets in *F*, then it must be contained (either on the boundary or within the interior) in the particular cell corresponding to *F* (see Observation 1). Combining this argument with the discussion in the preceding paragraph, one can conclude that *r* is bound to be present in *R* since set *R* encompasses all the 0-faces of the cells. Thus, the lemma follows.

Based on Lemma 2 and 3, it is possible to reduce our search space considerably from the entire arrangement *A* to only the cell boundaries. Moreover, from Lemma 3, it can be inferred that by only utilizing *R* as the search space, the maximum possible number of facets can be successfully included for all candidate retractions which may be present in the optimal solution. However, by ignoring the non-vertex points in the cell edges, the least possible retraction lengths may not be obtained. In order to minimize this error, the edges of the feasible retraction space polygons are sub-divided a priori (before computing the line segment intersections) so that no two neighboring vertices are more than ε apart from each other on any of the sides. Still the retraction lengths in the solution might be marginally greater than or equal to the optimum retraction lengths. However, Theorem 1 formalizes that such an error will be bounded.

Theorem 1: Let S^* be the optimal solution and let S' be a solution that has the same facet sets but each retraction length is increased by 0.5ε. Then the solution produced by the algorithm will be no worse than S'.

Proof: As Figure 3.10 points out, maximum error in retraction length occurs when the optimum solution lies at the mid-point of an edge and the solution explored by the algorithm by searching over the set *R*, corresponding to the same facet set is obviously one of the edge vertices. If the optimum retraction corresponds to any other non-vertex point on the edge, then the closer neighboring vertex will be considered by the algorithm. In such a situation, there are two possibilities. First, the algorithm will return this closer neighbor solution, resulting in a small error. This error in retraction length is, of course, bounded by half the maximum possible edge length, i.e. by 0.5ε. Second, the algorithm will find a better overall solution (involving different facet sets and retractions) than this solution.

From the above argument, it easily follows that since solution $S^{\prime}$ has identical facet sets as the optimum solution S^*, but each retraction length is increased by 0.5ε, the algorithm will not generate solution worse than $S^{\prime}$.



Cell
ΔY
Maximum error is equal to half the greatest length possible for any cell edge
Optimum retraction
ΔX
Retraction returned by our algorithm

Figure 3.10: Maximum error in retraction length

Choosing ε to be equal to 1 mm, there is very little qualitative difference between the solution generated by the algorithm and the optimum one. Practically this error has no effect, since a discrepancy of 0.5 mm in the retraction length hardly matters due to use of fast actuators attached to side actions. Thus, this error is insignificant.

After generating this set *R*, in order to make the optimization (e.g., depth-first branch and bound) algorithm more efficient, it is important to eliminate all the dominated retractions that are never going to produce the optimal solution. Then a test is carried out to determine whether every remaining retraction is a feasible one. The approach for pruning such redundant retractions, testing feasibility, and then arranging the non-dominated, feasible retractions and their associated retractable facets in the form of efficient data structures is explained in Section 5.1.

Finally, the retractions are treated as nodes to construct a search tree in a particular manner, described in details in Section 5.2. Certain heuristic rules have been developed to ensure that branching of the tree at any level is restricted within manageable proportions. The maximum depth of the tree will also be restricted to 3 or 4 in most practical cases. A depth-first branch and bound strategy is then employed on this search tree to obtain the optimum set of undercut regions. These regions are then swept along the horizontal translation vectors corresponding to the retractions and additional planar patches are inserted to form compact side action solids.

Chapter 4

COMPUTING CANDIDATE RETRACTION SPACES

From the discussion in the previous chapter, it is clear that first of all the candidate retraction space needs to be constructed for every individual undercut facet. In order to describe the steps for doing that, certain definitions have been introduced in Section 4.1. The overall procedure can be decomposed into two stages. Initially, an candidate translation space needs to be computed on the retraction plane such that a single horizontal translation vector can pull the facet there without being obstructed by any other facet lying on the way. The algorithm for computing this has been explained in details in Section 4.2. To ensure that the horizontally translated undercut facet can also be pulled vertically, it is necessary to construct another candidate translation space on the retraction plane. Eventually, the two candidate translation spaces have to be concatenated to obtain a combined, candidate retraction space. All these steps have been described in Section 4.3.

4.1 Related Definitions

Collision polyhedron for a facet f with respect to another facet f' is defined as the set of points in 3D translation space such that if corresponding translations are applied to f, then the translated f intersects with f'.

Collision polygon for a facet f with respect to another facet f' is defined as the set of points in 2D translation space such that if corresponding translations are applied to f, then the translated f intersects with f'.

Sweep-based collision polygon for a facet f with respect to another facet $f^{/}$ is defined as the set of points in 2D translation space such that if corresponding translations $\vec{t}$ are applied to *f*, the swept polygon $P^{/}$ will intersect $f^{/}$, where $P^{/} = \{p + \xi \vec{t} : p \in f, 0 \le \xi \le 1\}$.

Collision free 2D translation space for a facet f is defined as the set of points in 2D translation space such that if corresponding translations $\vec{t}$ are applied to *f*, the swept polygon $P^{/}$ (defined as before) will not intersect with any of the collision facets $f^{/}$.

Inaccessible polygon for a facet f refers to the set of points in 2D translation space such that if this translation is applied to *f*, then it is not accessible vertically upwards.

2D translation space for upward vertical accessibility for a facet f is defined as the set of points in 2D translation space such that if this translation is applied to *f*, then it can be released vertically upwards. Similarly, a 2D translation space for downward vertical accessibility can also be defined for facet *f*.

4.2 Computing Collision-free 2D Translation Spaces

Firstly all the potential facets capable of causing obstruction need to be identified. Since the translation vector in this case is restricted to lie in horizontal plane, it makes sense to consider all the facets lying partially or completely within the z-range of the undercut facet under consideration as potential obstacles. Z-range refers to the 3D space bounded by the maximum and the minimum z coordinate values of the facet vertices. In case a facet does not lie completely within the z-range, only the part of it lying inside is considered. This involves truncation of triangular facets to form convex polygons of 3, 4 or 5 sides.

According to Aronov et al [Aron97], 3D free configuration space *FP* of a convex "robot" *B* moving in space occupied by k' obstacles $\{A_1, \ldots, A_{k'}\}$ is given as the complement *C* of *U*, where $U = \bigcup_{i=1}^{k'} P_i$ be the union of the so-called expanded obstacles P_i. Here P_i is the Minkowski sum (illustrated in Figure 4.1) of A_i and $-B$, for $i = 1,\ldots, k'$.

$$P_i = A_i \oplus (-B) = \{a - b : a \in A_i, b \in B\} \tag{4.1}$$

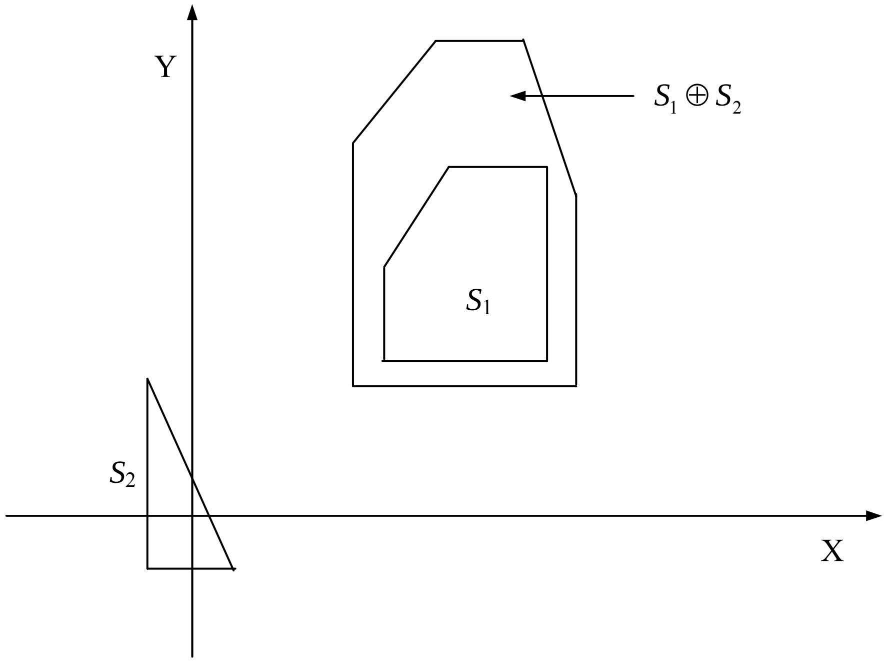


Figure 4.1: Minkowski sum of two convex polygons

In this case, replacing *B* by the undercut facet under consideration and A_i^s by the k' facets falling within its z-range, set of collision polyhedrons are obtained. Since this Minkowski sum is computed easily by obtaining the convex hull of the vector differences of each pair of vertices, total process has $O(k'm' \log m')$ worst-case asymptotic time complexity, where m' is the maximum number of vertices in each of the expanded obstacles. The collision polyhedrons are then intersected by a horizontal plane located at $\Delta z = 0$ to obtain collision polygons in the retraction plane.

However, the fact that the facet must be able to reach this candidate space by means of a single translation vector also needs to be taken into account here. That is why, mere construction of Minkowski polyhedrons and then intersection with a horizontal plane do not give the final obstructed space.

The hard shadows (shown in Figure 4.2) of all the $k^{/}$ collision polygons need to be computed in order to determine the sweep-based collision polygons. Alternatively, same thing is done by plane sweeping the collision polygons until they reach the retraction plane boundaries. Assuming that $n^{/}$ is the maximum number of vertices present in a collision polygon, this has a worst-case asymptotic time complexity of O($k^{/}n^{/}$ log $n^{/}$). Lastly, the collision free 2D translation space is obtained by subtracting the union of the forbidden spaces from the bounded retraction plane. Each union and subtraction operation again takes O($n^{//}$ log $n^{//}$ + $I^{/}$ log $n^{//}$) time [deBe00], where $n^{//}$ is the total number of vertices of two polygons and $I^{/}$ is the complexity of the resulting polygon. All the steps mentioned so far have explained formally in the following algorithm COMPUTE-MINKOWSKI-SINGLE-TRANSLATION-SPACE and illustrated in Figure 4.3.

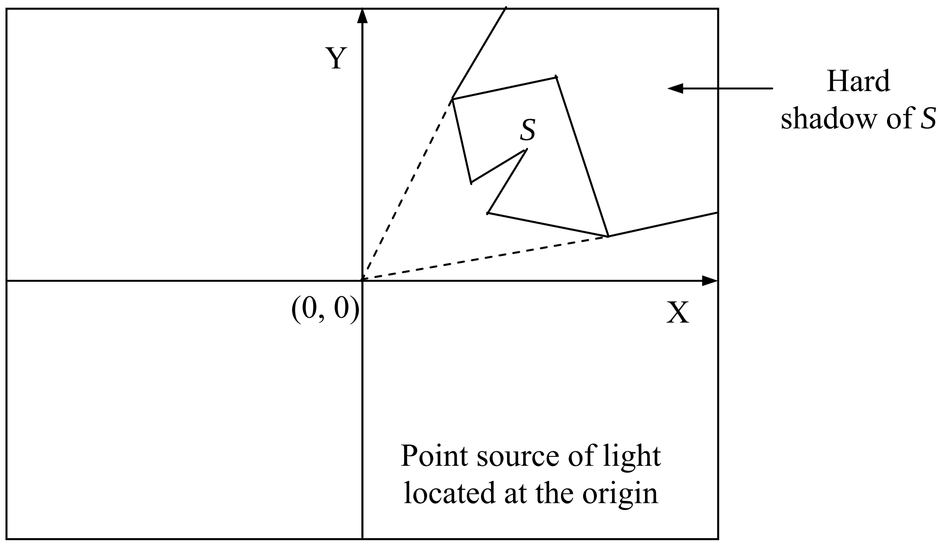
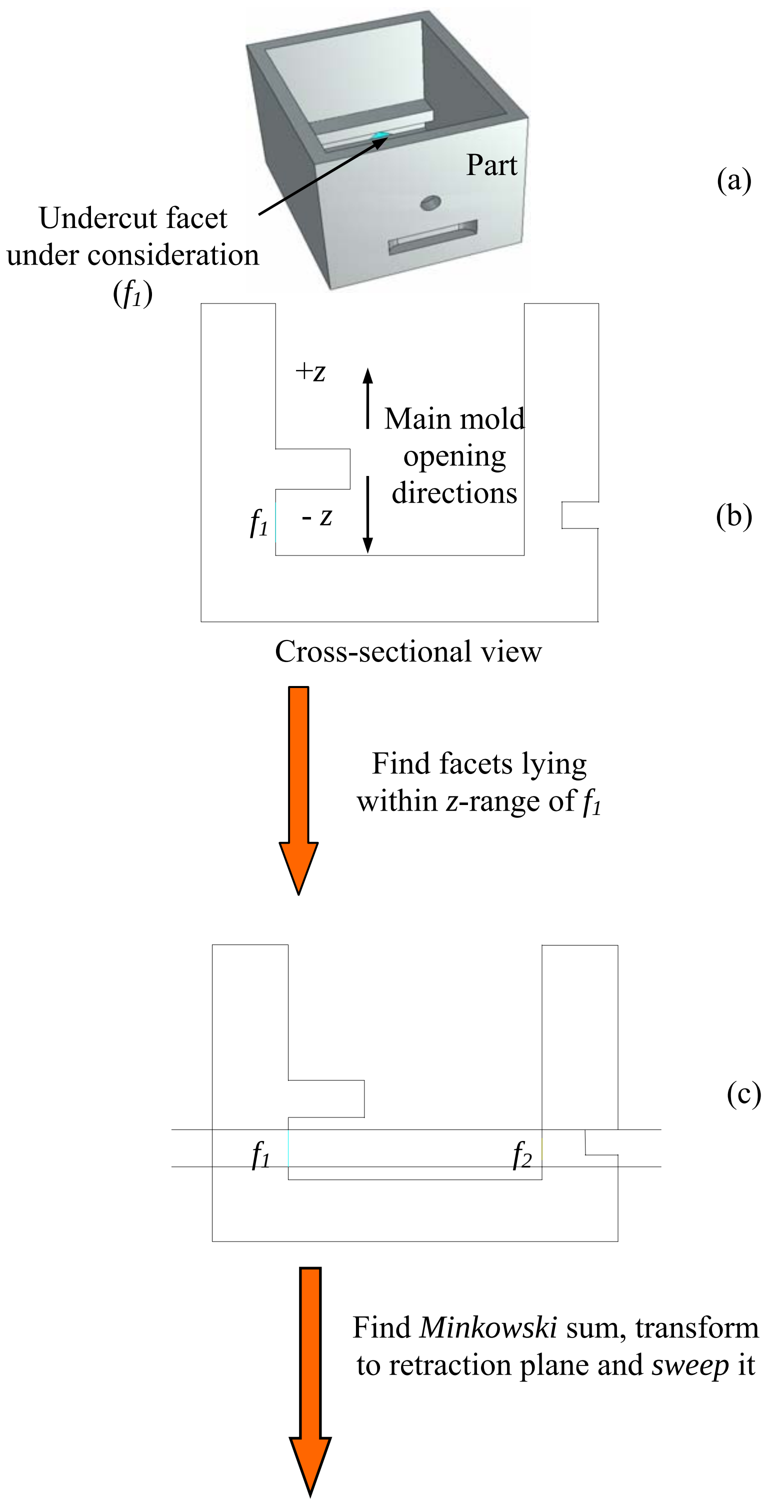


Figure 4.2: Hard shadow of a polygon


Part
Undercut facet
under consideration
(f_1)
(a)
+z
Main mold
opening
directions
f_1
- z
(b)
Cross-sectional view
Find facets lying
within *z*-range of f_1
f_1
f_2
(c)
Find *Minkowski* sum, transform
to retraction plane and *sweep* it

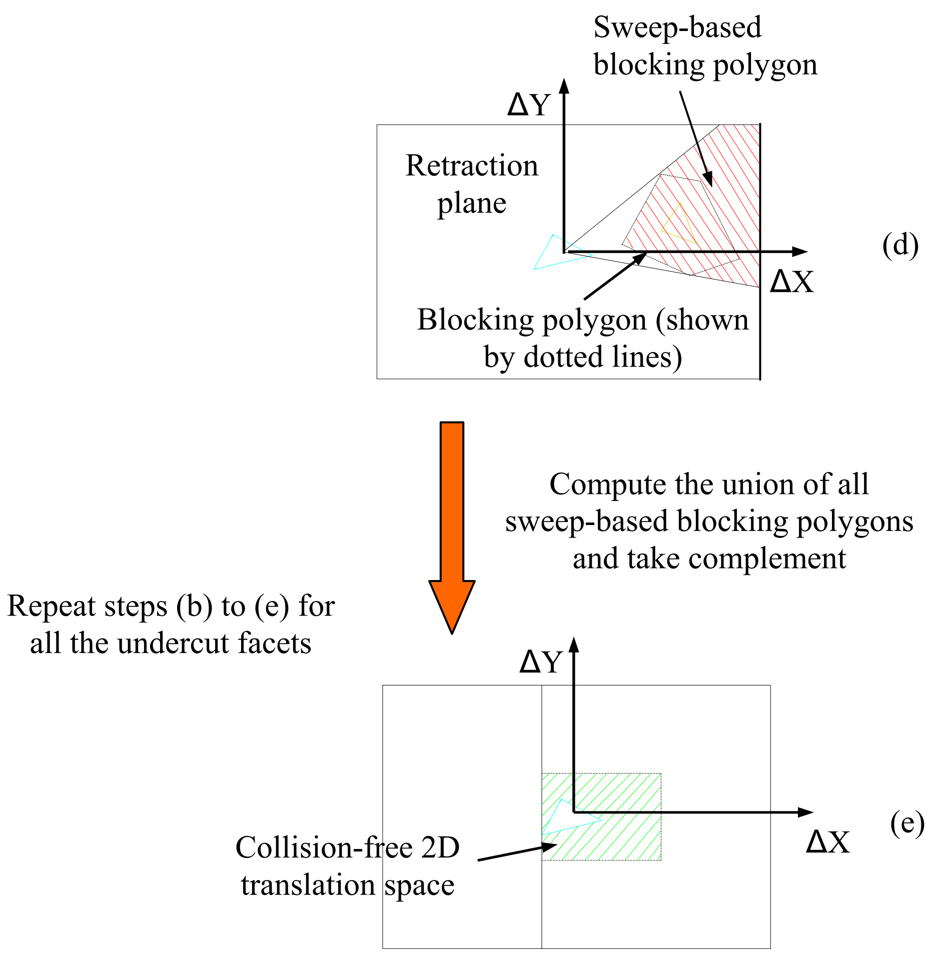

Sweep-based blocking polygon
ΔY
Retraction plane
(d)
ΔX
Blocking polygon (shown by dotted lines)
Compute the union of all sweep-based blocking polygons and take complement
Repeat steps (b) to (e) for all the undercut facets
ΔY
(e)
ΔX
Collision-free 2D translation space

Figure 4.3: Constructing collision-free 2D translation space

Algorithm: COMPUTE-MINKOWSKI-SINGLE-TRANSLATION-SPACE

Input:

- N facets representing the part P and its mold enclosure B.
- A set of n undercut facets $F_u = \{f_1,\ldots,f_n\}$.

Output:

- A set of n collision free 2D translation spaces $C_1 = \{c_1,\ldots,c_n\} \in R^2$, where c_i corresponds to undercut facet f_i.

Steps:

1. **For** every f_i in F_u, **do**

```
i. Create a z-range by considering the overall vertical stretch or extent of $f_i$.
ii. Form a new set $F_s = \{f_1,\ldots,f_m\}$, such that all triangles $f_j \in F_s$ (each $f_j$ is a sub-
    set of a particular facet forming the part) have vertical ($z$) extents situated
    partially or completely within the $z$-range defined for $f_i$ in the preceding step i.
    If a facet lies completely in the z-range, then the whole of it is included.
    Else only that part of facet, where the vertical extent lies within the $z$-range, is
    included and remaining part is truncated.
iii. Set index $j = 1$.
iv. While $j <= m$, do
        a) Determine the collision polyhedron for $f_i$ with respect to $f_j$
           $p_j = f_j \oplus (-f_i)$, $f_j \in F_s$, by calling the function COMPUTE-
           MINKOWSKI-SUM ($f_j$, $f_i$).
        b) Find the sweep-based collision polygon for $f_i$ with respect to $f_j$,
           namely $s_j$, by calling the function COMPUTE-SINGLE-
           TRANSLATION ($B$, $p_j$, $f_i$).
        c) Increment $j$ by 1.
v. Compute the union $m_i$ of all the 2-D convex polyhedra $s_j$;
                    $m_i = \cup_{j=1}^{m} s_j$ .
vi. Determine $c_i$ by taking the complement of $m_i$, i.e. $c_i = m_i^c$ .
2. Output $C_1 = \{c_1,\ldots,c_n\}$.
```

Function: COMPUTE-MINKOWSKI-SUM

Input:

- Two facets f_1 and f_2.

Output:

- A 3-D convex polyhedron p_1.

Steps:

1. **For** each pair v, w of vertices, with $v \in f_1$ and $w \in f_2$, compute $v - w$.
2. Find out the convex hull p_1 of the all these differences (3 for triangular facets) obtained in step 1.
3. Output p_1.

Function: COMPUTE-SINGLE-TRANSLATION

Input:

- Mold enclosure B.
- A 3-D convex polyhedron s_1 and a facet f_1.

Output:

- A 2-D convex polyhedron $s_1^{/}$, lying in the horizontal plane.

Steps:

1. Transform s_1 into a convex polygon r_1 lying in 2-D translation (Δ XY) space by intersecting s_1 with the horizontal plane, lying at $\Delta z = 0$.
2. Compute the hard shadow of r_1 assuming there is a point source of light situated at the origin and label this shadow zone as b_1. This zone is bound by the mold enclosure. Alternatively b_1 can also be obtained by plane-sweeping r_1 until it intersects the boundaries of B.
3. Concatenate the two regions r_1 and b_1 to obtain another convex polyhedron $s_1^{/}$; $s_1^{/} = r_1 \cup b_1$
4. Output $s_1^{/}$.

4.3 Computing 2D Translation Spaces for Vertical Accessibility

To compute this space, the entire mold free space is first sub-divided into cubical voxels. The length of each side of the cube is of the order of the dimension of the triangular facets. The accessibility of all the voxels lying within the *z*-range for each facet in +z direction is determined and all the inaccessible voxels are grouped together. This is done by checking whether the swept volume of each voxel intersects with the part *P*. A voxel is reported as inaccessible even if there is partial intersection. These grouped voxels are transformed into a 2D plane by intersecting with a horizontal plane at *z* equal to the height of the facet centroid to form inaccessible polygons. Basically, any point within the *z*-extent of the facet can be chosen since its collision-free horizontal translation is already taken care of. Now, the Minkowski sum between all the inaccessible polygons and the reflection of the facet (also intersected by the same horizontal plane) needs to be computed and the final product subtracted from the retraction plane to obtain the exact 2D translation space for upward vertical accessibility.

Although the inaccessible polygon may not be convex, convex decomposition is easily carried out before computing Minkowski sums by identifying concave corners and constructing additional edges parallel to either of the two axes (X or Y), since the original edges will always be axis-aligned. For any polygon having n_i vertices, this is done optimally in $O(n_i \log \log n_i)$ time by using the maximum non-intersecting chords method described in [Liou90]. Finally, intersection of the collision-free 2D translation space and the 2D translation space for upward vertical accessibility is taken to identify the candidate core retraction space for every individual undercut facet. A similar approach is adopted to compute the candidate cavity retraction space by considering accessibility of the voxels in –*z* direction. The steps are schematically shown in Figure 4.4. This step results in two different candidate retraction spaces: one for +z direction (core) and a second one for –z direction (cavity) and have been explained below in the algorithm COMPUTE-CANDIDATE-RETRACTION-SPACES. In the current implementation, steps described in Sections 5.1 and 5.2 are executed for both of these spaces separately.

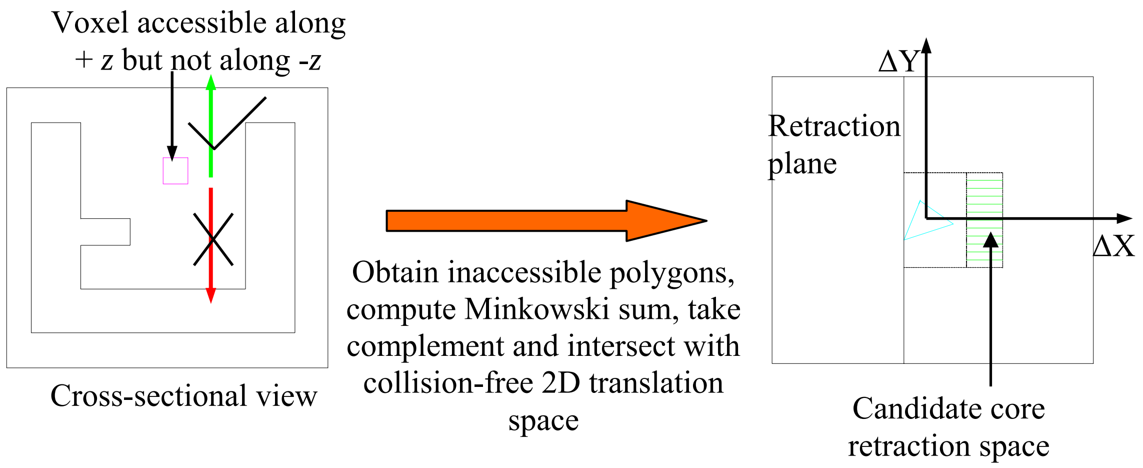

Voxel accessible along + z but not along -z
Cross-sectional view
Obtain inaccessible polygons, compute Minkowski sum, take complement and intersect with collision-free 2D translation space
ΔY
Retraction plane
ΔX
Candidate core retraction space

Figure 4.4: Constructing candidate core retraction space

Algorithm: COMPUTE-CANDIDATE-RETRACTION-SPACES

Input:

- *N* facets representing the part *P* and its mold enclosure *B*.
- Set of *n* undercut facets $F_u = \{f_1,\ldots,f_n\}$.
- Set of *n* collision free 2D translation spaces $C_1 = \{c_1,\ldots,c_n\}$ obtained as an output of the algorithm COMPUTE-MINKOWSKI-SINGLE-TRANSLATION-SPACE.

Output:

- Set of *n* candidate core retraction spaces $C_1^{/} = \{c_1^{/},\ldots,c_n^{/}\}$.
- Set of *n* candidate cavity retraction spaces $C_1^{//} = \{c_1^{//},\ldots,c_n^{//}\}$.

Steps:

```
1. Compute mold free space $MS$ by taking the regularized Boolean difference between the mold enclosure and the part, i.e. $MS = B -^* P$.
2. Sub-divide $MS$ into a large number of cubical voxels such that length of all the cubes is identical and is of the order of the part facet dimension. Let $V = \{v_1,\ldots,v_q\}$ represent the set of voxels.
3. Create two empty sets $C_1^{/}$ and $C_1^{//}$.
4. For every $v_i \in V$, do
    i. Create two swept volumes (solids) $S_1$ and $S_2$ by sweeping $v_i$ till it reaches the boundaries of $B$ in $+z$ and $-z$ directions respectively.
    ii. Test whether $S_1$ and $S_2$ intersects with the polyhedral part $P$. If $S_1$ intersects $P$, then assign a label 1 to $v_i$. On the other hand, if $S_2$ intersects $P$, then assign a label 2 to $v_i$.
5. While the set $V$ is not empty, do
    i. Start randomly from any seed voxel $v_i$ which has a label 1 and set $i = 0$.
    ii. Apply seed fill technique to find all other neighboring voxels sharing the same label. Unionize these voxels to form to a polyhedron $p_i^{/}$ ($i = i + 1$) and then delete all these voxels from $V$.
    iii. Start with any other seed voxel having label 1 randomly (as long as one exists) and follow previous step ii.
    iv. Start randomly from any voxel $v_j$ that has a label 2 and follow steps ii. and iii. to form polyhedrons $p_i^{//}$ by grouping together all the voxels carrying label 2.
6. For every $f_i \in F_u$, do
```

i. Transform all the polyhedrons $p_i^{/}$ and $p_i^{//}$ to retraction plane by intersecting them with the horizontal plane *z* equal to the *z*-coordinate of the centroid of f_i. This operation results in the formation of two sets of inaccessible polygons $po^{/}$ and $po^{//}$, corresponding to $p_i^{/}$ and $p_i^{//}$ respectively.

ii. Decompose each rectilinear polygon present in the two sets $po^{/}$ and $po^{//}$ into minimum number of convex pieces; then compute Minkowski sum between every such convex polygon and $-(f_i(0,0))$ and unionize all the Minkowski polygons for the sets individually. Finally, take the complement of both the unionized polygons separately to form 2D translation spaces for upward and downward vertical accessibility for facet f_i respectively. Let these two spaces be denoted as u_i and d_i.

iii. Form the candidate core retraction space $c_i^{/}$ corresponding to f_i by intersecting c_i and u_i, i.e. $c_i^{/} = c_i \cap u_i$. Insert $c_i^{/}$ in the set $C_1^{/}$.

iv. Similarly insert an element $c_i^{//}$ in the set $C_1^{//}$ by using d_i instead of u_i.

7. Output $C_1^{/}$ and $C_1^{//}$.

Using the lemmas and theorem presented in Section 3.4, the candidate retraction spaces thus obtained will be discretized to obtain a set of candidate retractions. Next a set of non-dominated retractions will be identified from this discretized set. Each non-dominated retraction will then be tested to determine whether it is a feasible retraction. Finally, all the feasible retractions will be arranged in the form of data structures so that they can be queried efficiently at the time of generating the optimal solution from the feasible search space. These steps have been discussed in details in the next chapter along with the results on a number of test parts.

Chapter 5

FORMING SIDE ACTIONS

The final goal of the current work is to generate a set of side actions which can be actuated simultaneously and independently of one another, such that each of them can mold a portion of the undercut region. This is equivalent to combining a set of undercut facets together when they share at least one common feasible retraction. At the same time, the given cost function needs to be minimized (see Sections 3.2 and 3.3). Hence, this grouping has to satisfy global optimum (or near-optimum) criterion. Sampling of the candidate retraction spaces computed in the previous chapter to obtain a discrete set of feasible retractions is explained in Section 5.1. Global search strategy has been described in Section 5.2. Finally the implementation details, including a discussion on the computational experiments, summary of results and figures of generated side action solids have been presented in Section 5.3.

5.1 Computing Discrete Set of Feasible Retractions

As discussed in Section 3.4, the aim is to obtain the discrete set of all possible retractions. Firstly, edge sub-division is carried out on all the candidate retraction space polygons if necessary and then all the line segment intersection points are computed. The corner points of all the retraction space polygons are also retained.

Next the set of retractions needs to be pruned to obtain a so-called non-dominated set. Such a set will only consist of those retractions which are better than any other in the entire set with respect to either the retraction length or the number of associated facets.

This is done by first sorting all the retractions in order of increasing retraction lengths and is completed in $O(m \log m)$ time [Corm01], where m is the total number of retractions. Then all the retractions that have identical retraction lengths are listed and again sorted in order of their number of retractable facets. All the retractions that have lesser number of associated facets than the current maximum value are deleted. Initially, this current maximum is equal to zero and is progressively updated as one retraction length bucket is considered after another. Each deletion operation takes constant time and a binary searching is implemented in $O(\log m)$ time.

After obtaining the non-dominated set, it needs to be tested whether each retraction is a feasible one, and if so, for which all retractable facets. A particular retraction may be a feasible one for certain facet(s) in its retractable facet set, whereas, it may not be so for the remaining facets in that set. Hence, this check has to be carried out individually for every retractable facet associated with all the q non-dominated retractions. Actual checking is quite straightforward; for a given undercut facet and a retraction, the minimum length of the side action solid is computed easily as the 2D translation space for vertical accessibility is known to us (see Chapter 4). Then, this minimum length has to be vectorially added to the horizontal translation (along the same direction) already associated with the given retraction. If the final retraction lies inside the candidate retraction space for the undercut facet, then it is a feasible retraction; otherwise not. Whenever the answer is no, that particular undercut facet is removed from the set of retractable facets. At the end, if any undercut facet has no feasible retraction associated with it, then it can be immediately concluded that the given undercut region is non-moldable and the part needs to be re-designed. In that case, the overall algorithm terminates without entering into the optimization phase. This eliminates the need for costly simulation-based post-processing operations to deal with infeasible side action designs.

A look-up table *LT* is then constructed to satisfy another output requirement: non-intersection of generated side action solids. It needs to be ensured that different solids formed by sweeping individual undercut facets along the horizontal translational vectors associated with corresponding retractions do not intersect each other. If any two solids intersect for any two given feasible retractions, those retractions cannot occur in the same solution set. Any entry in the table has a value 1 if the two retractions corresponding to the particular row and column under consideration can be present in the same solution. Otherwise, it is assigned a value 0. Such an intersection check only needs to be carried out for the facets that are not present in both the retractable sets and lie within the z-range (defined in Section 4.2) of each other. Although this pre-processing step is computationally intensive, it ensures the correctness of our solution from the actual application (molding operation) point of view and saves considerable time in the optimization stage. For example, for the look-up table shown in Figure 5.1, side actions corresponding to 1^{st} and *q*th retractions can be present in the same solution set, whereas 2^{nd} and *q*th retractions cannot be included in the same output set.

	1st retraction	2nd retraction	• • •	*q*th retraction
1st retraction	NA	0	• • •	1
2nd retraction	0	NA	• • •	0
. . .	. . .	. . .	. . .	. . .
*q*th retraction	1	0	• • •	NA

Figure 5.1: Look-up table to ensure non-selection of intersecting side actions in the output set (NA means non-applicable)

Once all the non-redundant feasible retractions have been computed, a tree is constructed to represent the feasible solution space. In order to do that, the undercut facets have to be sorted in a particular way, so that the process of placing the nodes gets facilitated. A heap H is built for the n undercut facets in linear ($O(n)$) time [Corm01]. This is going to arrange the facets depending upon their minimum retraction lengths (with maximum preference being given to those having maximum values of minimum retraction lengths) and break ties on the basis of lesser number of associated retractions. This heap is used as an efficient priority queue. Extracting the most critical or highest prioritized facet can be easily done in $O(\log n)$ time. Similarly after a vertex has been considered, deletion of each corresponding facet is done in $O(\log n)$ time, so that overall process takes less than O(n log n) time even in the worst-case (since all facets can never be deleted from the heap).

Moreover, a set of linked lists LL = {ll_1, …, ll_n}is used so that access from undercut facets to retractions can be done easily in constant time (shown in Figure 5.2). For a particular element in the linked list, i.e. an undercut facet, all the associated retractions are maintained in a sorted order depending upon their lengths, with highest preference being given to the one(s) having least magnitude. This is utilized while placing the actual nodes in the search tree. All these data structures are created so that queries become efficient while traversing the search tree to obtain an optimal solution to the undercut region grouping problem. They ensure that no more than O(n log n) computations need to be performed at any node in any level of the tree, instead of the usual O(n^2) calculations necessary if exhaustive comparisons and enumerations are carried out.

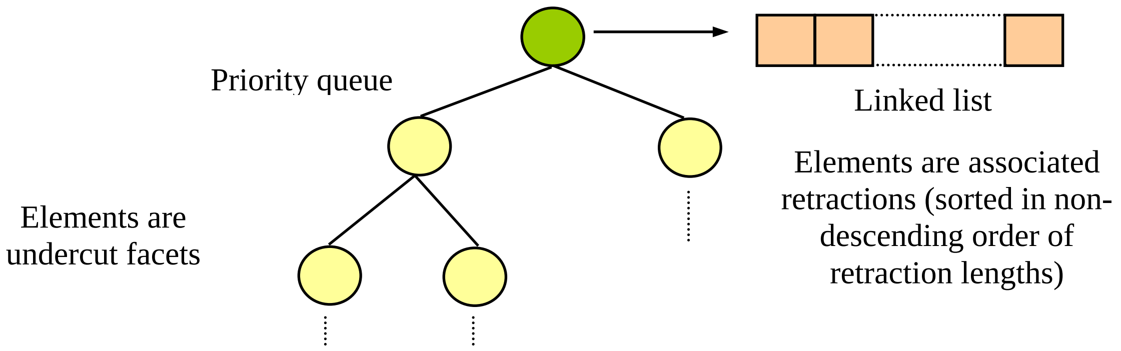

Priority queue
Elements are undercut facets
Linked list
Elements are associated retractions (sorted in non-descending order of retraction lengths)

Figure 5.2: Efficient arrangement of undercut facets and retractions

5.2 Global Optimum Search Algorithm

A depth-first branch and bound algorithm is used to determine the optimal set of undercut regions. This requires employment of intelligent heuristics to quickly steer search to a good initial solution, limit branching, and prune as many search paths as possible. The notion of a *bottleneck* facet plays a key role in realizing these heuristics.

It may be observed that certain facets become the main bottlenecks in generating a solution. These facets have very limited number of feasible solutions and they impose constraints on the goodness of the overall solution. The overall solution has to address these facets and hence it is desirable to process these facets first. These facets belong to the category such that the minimum retraction length for them is highest among all the facets. On top of this, they must have rather narrow feasible retraction spaces, which, in turn, will be reflected in the fact that they will have smaller number of retractions associated with them. Bottleneck facets for a given part are shown in Figure 5.3.

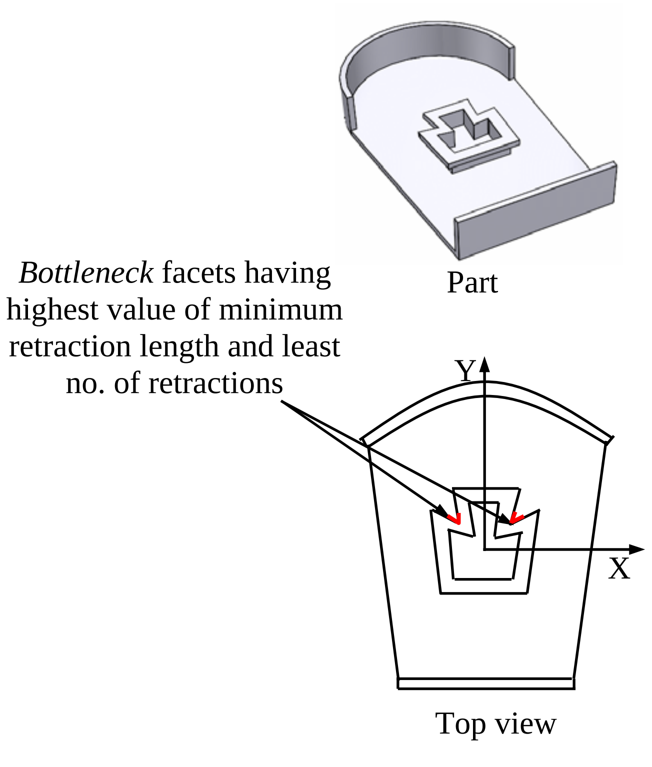

Part
Bottleneck facets having highest value of minimum retraction length and least no. of retractions
Y
X
Top view

Figure 5.3: Bottleneck facets

Once bottleneck facets have been identified, construction of search space is started with any one of them. An empty node is created as the root node and all the retractions associated with the chosen bottleneck facet (highest priority element in our heap) are placed as top level nodes in the search space. Since retractions are accessed directly using the linked list, they will be placed in order of increasing retraction lengths. If a particular retraction is considered, its associated retractable facets form the first undercut region. In the next level, one needs to determine the bottleneck facets among the remaining ones and place the retractions attached to any one of them as nodes. This proceed is repeated until all facets are covered. Of course, one needs to be careful not to include two retractions such that the corresponding side actions intersect with each other in the final solution. Such a path, if encountered, will be termed as infeasible (by using our look-up table) and will be pruned. If some of the retractable facets associated with a retraction are already covered then they will not be added to this solution.

Track is kept of the current best solution. If during the search, cost of a partial solution exceeds the cost of current best solution, then this path is pruned. If during the search a better solution than the current best solution is found then the current best solution is updated. However, improved forward-looking cost bounds can be easily derived as follows to prune more paths. Let C^* be the current optimum solution and C_N be the partial cost incurred so far along a particular path, after including N nodes (retractions). Then a path is pruned if it can be said certainly that C_{N+1} will be greater than or equal to C^*. Mathematically this pruning condition is equivalent to:

$$C^* - C_N < C^/ + \gamma\left(l_{\min}^g\right)^k + \chi l_{\min}^g \qquad (5.1)$$

where $l_{\min}^g$ is the minimum retraction length in the set of feasible retractions. This is pre-computed once before starting depth-first branch and bound. This equation follows from the fact that whenever another node is introduced, an additional side action is included which will, of course, have a retraction length greater than the minimum length possible.

However, a tighter bound can be obtained as well. This can be mathematically stated as:

$$C^* - C_N < C^/ + \gamma\left(l^r_{\min}\right)^k + \chi l^r_{\min} \tag{5.2}$$

where $l^r_{\min}$ is the minimum retraction length for all the remaining undercut facets that have not been included in the current partial solution so far. Although this will prune larger number of nodes in the search space, it entails additional run-time computations as one needs to look up for the minimum value among all the elements placed at the beginning of the linked lists associated with all the undercut facets present in the updated priority queue.

Search can terminate in two ways. Firstly, it terminates when all promising nodes have been explored. In this case it produces a solution that is very close to the optimal solution. Secondly, it stops when the user specified time limit has been exceeded. In this case search returns the current best solution.

The bottleneck facet scheme restricts the amount of branching (or number of nodes) at a particular search level. Now coming to the tree depth issue, it is rare to find parts in which a connected undercut region requires more than three side actions. Usually cost function parameters are such that solutions involving *N* + 1 side actions are more expensive than a solution involving *N* side actions. Our heuristics quickly locate feasible solutions and once a feasible solution has been found, very few nodes are explored at the next depth level. This characteristic ensures that tree depth is limited. For example, once a solution involving three side actions has been found, very few solutions involving four side actions appear promising enough to try. Solutions involving five side actions are virtually never tried in such a case. Hence, for virtually all practical parts in which a single connected undercut needs to be partitioned into three undercut regions one should be able to find optimal solutions in a reasonable amount of time. As with any depth first branch and bound algorithm, the time complexity increases exponentially with the depth of the search tree. However, since practical cases involving more than three side actions are not expected for a single undercut region, the exponential growth is not much of a concern in this particular application.

Once all the regions have been generated, they are swept along the associated horizontal translation vectors (corresponding to the retractions) and additional patches are included at the top, bottom and the arrow-tip end of the vector to form a compact, 2-manifold solid. This is illustrated in Figure 5.4. These capping patches consist of planar faces only. The boundaries of these solids are then triangulated to represent them in a faceted format and they form the desired set of side actions. All these steps are described in the following algorithm called GENERATE-SIDE-ACTIONS. Only the simplest bounding strategy has been stated in the algorithm below. However, any of the other two bounding strategies can be used by slightly modifying Step 5 a iv.

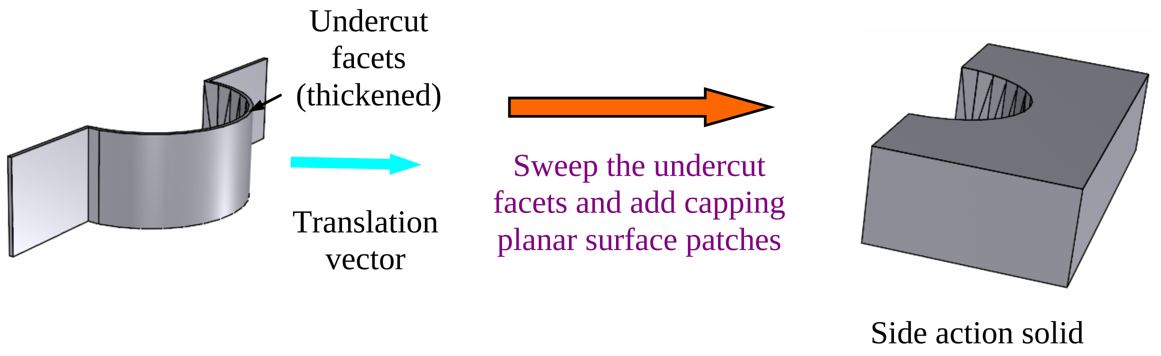

Undercut facets (thickened)
Translation vector
Sweep the undercut facets and add capping planar surface patches
Side action solid

Figure 5.4: Sweeping undercut facets to form side action solids

Algorithm: GENERATE-SIDE-ACTIONS

Input:

- Set of n undercut facets $F_u = \{f_1, \ldots, f_n\}$.
- Collection of p sets $\{\{\{f_{11}, \ldots, f_{n1}\}, l_1\}, \ldots, \{f_{1p}, \ldots, f_{np}\}, l_p\}\}$, each associated with a particular non-dominated retraction. Here, all $f_{ij} \in F_u$ (though may be repeated in different sets) and all l_i^s are retraction lengths. The first entity in every set is the set of retractable facets for that particular retraction.
- Look-up table LT, heap H and set of linked lists LL created during the pre-processing operations described in Section 5.1.
- Maximum permissible time t_{max} in seconds.

Output:

- One or more SAS $a_i = \{s_i, F_i, (x_i, y_i), T_i\}$ (as defined in Section 3.2).

Steps:

1. Initialize a tree T by creating an empty root node.
2. Initialize C_{min} to be positive infinity (very high value).
3. Populate the nodes in the next level by extracting the highest priority facet from the heap H and inserting all its associated retractions in order of increasing retraction lengths. Set LL is utilized to obtain this sorted retraction set directly. Color all the nodes as white.
4. Initialize this second level as the current one.
5. **For** every node u in the current level **do**

    a. **If** u is colored white or grey **do**

i. **If** *u* does not have a child node as yet, color the node as grey. **Else if** all the children nodes are colored black, color this node as black as well and unless the current level is the second one, decrement it by one.

ii. Check whether the side action corresponding to the retraction present in *u* intersects with any of the other side actions associated with the nodes in this tree path by using the look-up table *LT* directly. **If** yes, then *prune* the path by coloring *u* as black and decrement the current level by one; moreover go to step b. directly. **If** no, then further test whether the retractable facet set associated with *u* has a non-null intersection with the facet sets of the other nodes in the path. **If** answer is yes, then delete the common facets from the facet set.

iii. Estimate the incremental cost C_{inc} by considering all the facets associated with the retraction present in node *u* as a new undercut region F_i. Equation (3.1) is used to obtain C_{inc}.

iv. **If** by including F_i, all the facets in F_u are covered, then terminate this particular tree path and color *u* as black and decrement the current level by one. Moreover, in this case if $C_{inc} < C_{min}$, update C_{min} as C_{inc} and construct a set of undercut regions by inserting all the F_i^s of this particular path (from second level downwards) along with their associated retractions (x_i, y_i). Each F_i will also have a value of T_i (+1 or -1) depending upon whether this algorithm is implemented for the feasible core or cavity retraction space.

v. **Else**

1. **If** $C_{inc} < C_{min}$, temporarily delete all the facets of the set F_i from the heap *H*, insert all the associated retractions of the facet currently having maximum

priority in H as the children nodes in the next level in sorted order, color them as white and increment the current level by one.

2. **Else** *prune* this particular tree path in exactly the same way as in step ii.

b. **Else** go the next node u in the current level.

6. Once the recursive depth-first search loop 4 terminates after all the paths in T have been explored or after the user-defined time limit t_{max} has been exceeded, generate all the side core sets a_i^s by applying algorithm GENERATE-SOLIDS on each F_i to obtain a solid s_i.
7. Output all the a_i^s.

Algorithm: GENERATE-SOLIDS

Input:

- Undercut region F_i obtained in step 5 a iv. of the algorithm GENERATE-SIDE-ACTIONS and its corresponding retraction (x_i, y_i).

Output:

- A compact, 2-manifold solid body s_i with a well-defined semi-analytical boundary.

Steps:

1. Sweep F_i along the translation vector defined by the retraction (x_i, y_i).
2. Add additional planar faces at the top, bottom and arrow-tip end of the vector defined in step 1. to form a solid s_i. It should meet all the requirements specified in the output.
3. Triangulate s_i to form an output file in .stl format which can later be used for display purposes.
4. Output s_i.

5.3 Implementation

All the algorithms were implemented in C++ using Visual Studio.NET 2003 in Windows XP Professional operating system. CGAL [Cgal04] version 3.0.1 was used as the geometric kernel to carry out all the basic operations like regularized Boolean, computing the convex hull, finding intersections by using plane sweep and so on. In order to speed up computation, all the vertex coordinates were first converted into integral values and then the Cartesian kernel with int number type was used. OpenGL was used for displaying purposes. CAD model for every part was triangulated and converted into .stl format which was then taken as input by the program. The main mold opening direction was inputted separately. A preprocessor program was written to recognize all undercut facets. All the programs were run in a Pentium M processor machine having 512 MB of RAM and processor clock speed of 1.6 GHz. The overall system architecture is schematically shown in Figure 5.5.

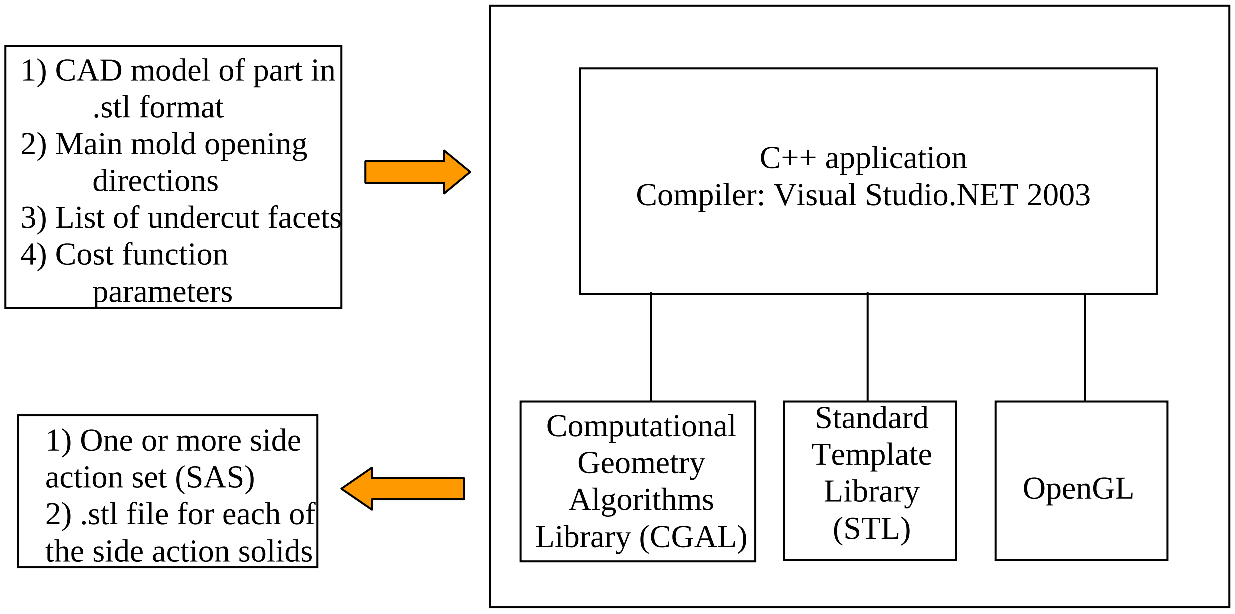

1) CAD model of part in .stl format
2) Main mold opening directions
3) List of undercut facets
4) Cost function parameters
C++ application
Compiler: Visual Studio.NET 2003
Computational Geometry Algorithms Library (CGAL)
Standard Template Library (STL)
OpenGL
1) One or more side action set (SAS)
2) .stl file for each of the side action solids

Figure 5.5: System architecture

The algorithm for computing 2D translation space for upward/downward vertical accessibility was implemented in a simplified form. Minkowski sums were not explicitly computed and the complement of the inaccessible polygons was directly taken. This change did not make any difference in practice since large numbers of accessible voxels occur together and a conservative approach was followed in classifying them as accessible or inaccessible.

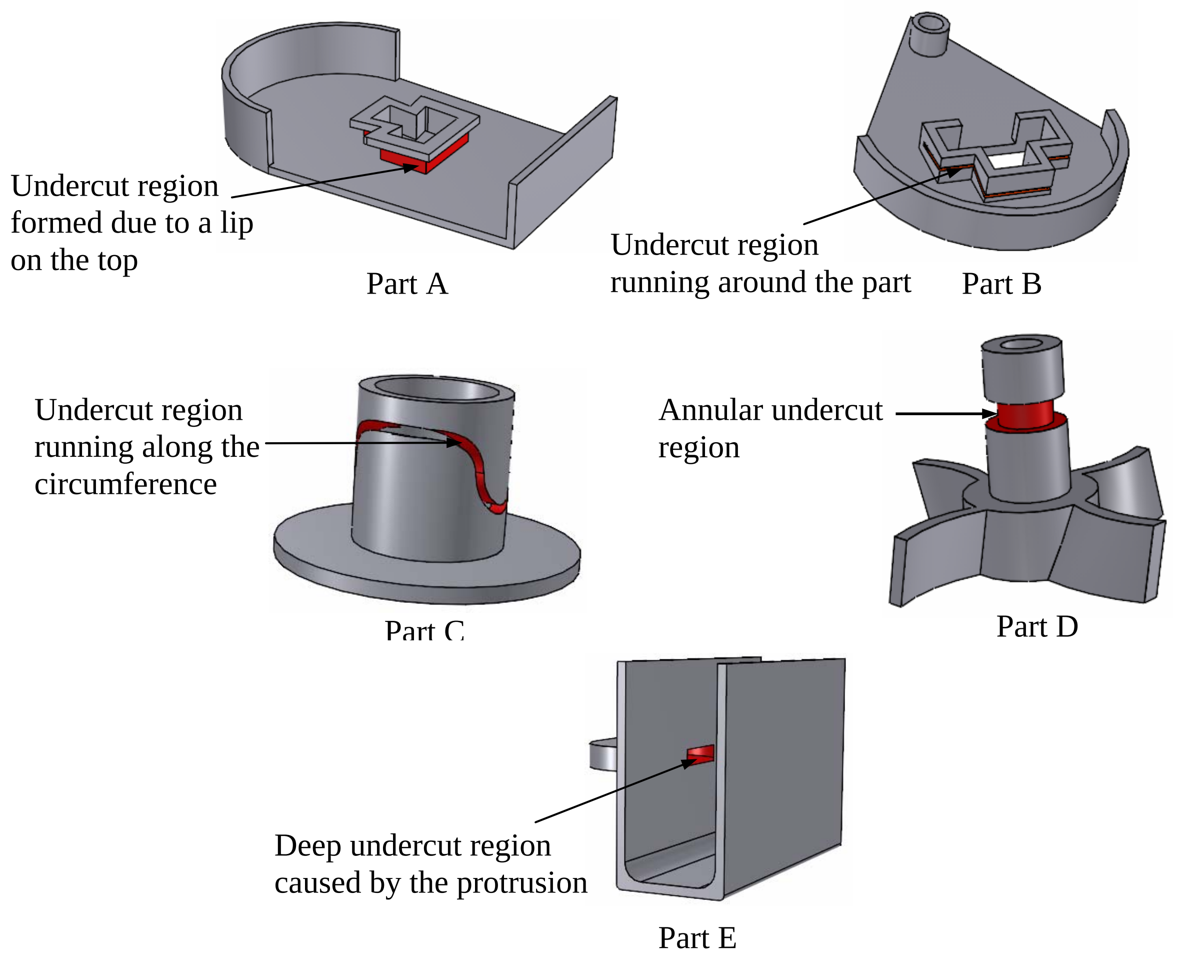

Undercut region formed due to a lip on the top
Part A
Undercut region running around the part
Part B
Undercut region running along the circumference
Part C
Annular undercut region
Part D
Deep undercut region caused by the protrusion
Part E

Figure 5.6: Five test parts

5.3.1 Computational Experiments

A series of computational experiments have been carried out on 5 parts (shown in Figure 5.6) to characterize the performance of the algorithms. Since time values cannot be compared across different parts, each part was faceted using four different levels of accuracy. The first two parts have same number of undercut facets in all the four models, while the total number of facets increases. However, both the number of undercut facets as well as the total number of facets increases across the four models in case of the latter two parts. Now comparisons with respect to computation time, number of nodes in search space and so on can be made for the same part having varying number of facets. Results of these experiments for γ = 200 (in \$/mm^2), k = 2, $C^{/}$ = 5,000 or 10,000 (in \$) depending upon whether it is a cavity or core retraction, and χ = 18.227 (in \$/mm) are shown in Table 1 below in Section 5.3.2. These values are based on a specific injection molding scenario. The second bounding strategy (defined by Equation (5.1)) is used in this case. The undercut region in part E is non-moldable and hence computation stops before entering the optimization stage. The side action solids generated for the other four test parts have been displayed in Figure 5.7 in Section 5.3.3.

Another series of computational experiments have been performed on the model containing highest number of facets for each of the first 4 test parts to estimate the relative performance of the three bounding strategies. This performance has been characterized on the basis of two parameters, namely, the number of pruned nodes and the overall computation time for depth-first branch and bound search. Results of these experiments have been summarized in Table 2 in Section 5.3.2.

5.3.2 Summary of Results

Certain basic trends are discernible from the values in the table. Of course, computation times for both the candidate retraction space and discrete set of feasible retractions increase as one goes for higher number of facets to represent the part. However, for parts A and B where the numbers of undercut facets remain same, although the overall numbers of facets increase, this trend is markedly different from parts C, D and E. The retraction space computation time increases linearly, whereas feasible retraction set calculation time remains more or less constant in the former case. In the latter case, candidate retraction space construction time increases almost quadratically, while the feasible retraction set computation time increases at a rate slightly greater than linear, but less than quadratic, indicating possibly a linear-logarithmic growth.

Computational results for the branch and bound algorithm (using second bounding cost function) reveal that the branching factor at any level in the search space tree does not change much with the increase in number of part facets. It increases only marginally, thereby demonstrating value of our heuristic rules on pruning the feasible retraction set, selecting the bottleneck facet etc. This clearly points out that the number of pruned, non-dominated retractions is not significantly affected by the part triangulation resolution, and hence, the number of nodes in the tree grows slowly in most practical cases. However, the overall computation time will increase somewhat if the parts are represented by higher number of facets. This is because of the fact that more time will be expended in carrying out all the operations in individual nodes. Based upon the given set of cost function parameter values, an optimal set of 2 or 3 side actions were generated for four sample parts in about 30-50s. This is a reasonably good performance and can serve as the foundation step towards our eventual goal of fully automatic side action design.

Table 5.1: Results of experiments to characterize algorithm performance

Part	Model	Total # of facets	# of undercut facets	Candidate retraction space computation time (in s)	Feasible retraction set generation time (in s)	Total # of nodes in search space (in million)	Average branching factor	Depth-first branch and bound computation time (in s)
A	# 1	224	36	3.5	5.2	13.4	99	12.0
	# 2	256	36	4.0	5.4	14.0	100	13.0
	# 3	336	36	4.8	5.9	15.6	102	15.2
	# 4	568	36	6.1	6.4	18.9	106	18.0
B	# 1	378	122	6.0	10.7	19.5	114	19.0
	# 2	570	122	7.4	11.1	19.8	115	19.1
	# 3	716	122	8.5	11.3	20.7	117	19.6
	# 4	882	122	10.0	11.4	21.9	119	20.2
C	# 1	414	175	5.9	12.5	0.36	150	1.1
	# 2	478	188	6.3	12.6	0.36	150	1.1
	# 3	576	194	7.1	12.8	0.37	151	1.2
	# 4	882	249	12.2	13.7	0.38	153	1.3
D	# 1	376	60	0.3	7.4	0.45	132	1.6
	# 2	814	122	1.0	9.8	0.46	134	1.7
	# 3	1324	156	2.1	12.9	0.49	137	2.0
	# 4	2002	218	4.4	20.2	0.51	139	2.4
E	#1	206	54	4.9	6.3	These steps have not been performed		
	#2	258	78	9.0	6.8			
	#3	282	94	11.1	7.1			
	#4	390	126	16.8	8.9			

Table 5.2: Results of experiments to compare bounding strategies

Part	Strategy	# of pruned nodes (in million)	Computation time (in s)
A	# 1	17.5	29.0
	# 2	18.2	18.0
	# 3	18.6	25.7
B	# 1	20.1	34.8
	# 2	21.0	20.2
	# 3	21.3	29.9
C	# 1	0.34	2.2
	# 2	0.36	1.3
	# 3	0.36	2.1
D	# 1	0.45	3.6
	# 2	0.47	2.4
	# 3	0.48	3.7

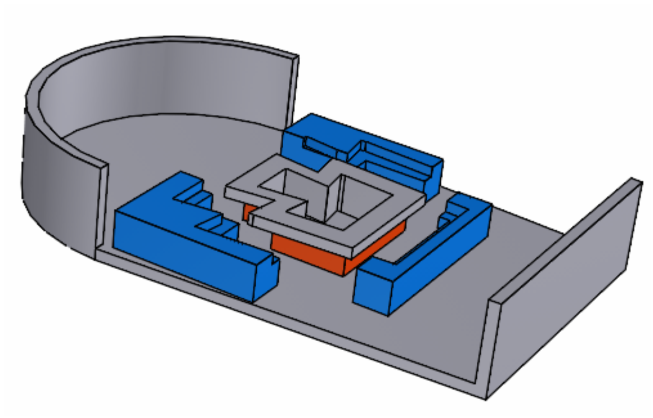
It can be easily seen from Table 2 that the second strategy consistently outperforms the other two for all the 4 test parts. Although few more search space nodes are pruned by applying the last strategy, it always results in greater computation time than the second one, and sometimes even more than the first one. This is because another level in the current path is indirectly explored before deciding not to proceed any further. Although nodes are not actually inserted in the next level, the priority queue is updated by removing the included undercut facets and then recovering the first element from all the associated linked lists to compute the minimum among them. Undoubtedly, this will prune more paths than either of the other two strategies. But already lot of computational time has been expended in obtaining this pruning cost. This turns out to be more than the time spent in exploring few additional paths.

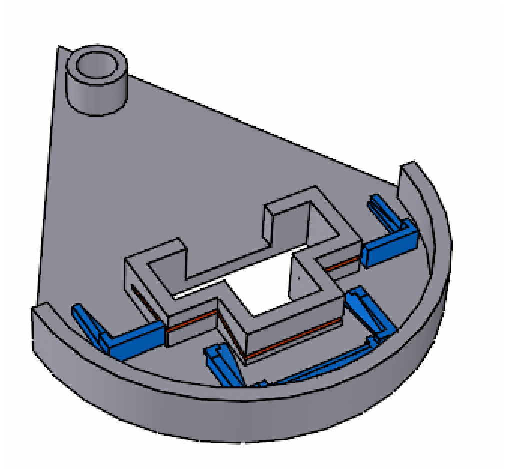
Such a situation arises in this case primarily because all industrial parts typically have 2 or 3 piece solutions. As the number of levels in our tree hardly goes beyond 3, the third strategy is not being to able to prune significantly higher number of nodes than the second one. Hence, such a tight bound may yield better results for parts requiring 4 or more side actions. However, such parts are hardly encountered in practice.

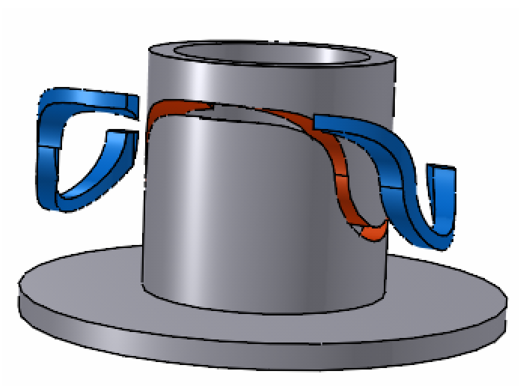
The second strategy also takes lesser computational time as compared the first (basic) one since it successfully prunes many more nodes in the search space without incurring any significant overhead at all. At every step, it just needs to be checked whether the difference between the current minimum cost and the partial cost is lesser than a value that has been computed once before starting depth-first branch and bound. This simple test is computationally inexpensive and yet very effective.

5.3.3 Generated Side Action Solids

Side action solids have been shown in blue for the first 4 test parts in a retracted state in Figure 5.7 for easy understanding of how they will be actuated in reality. The undercut facets have been colored red. It can be easily observed from the figures that in all the 4 cases, the undercut facets forming complex undercut regions have been combined into 2 or 3 groups. Then each such group has been swept along different directions by fixed translation vectors lying on the horizontal plane to form regular solid bodies. These solid bodies will eventually be actuated by suitable mechanisms such as sliders, hydraulic or pneumatic drives and so on to form the overall side action mechanism. The important point to be noted here is that the generated side actions never intersect with the part during its disengagement process. Moreover, after initial horizontal retraction, the part (or equivalently the side action) can be moved vertically upwards or downwards along one of the main mold opening directions without resulting in any intersection either. Thus, all the designs are correct and have been obtained automatically without any manual intervention.


A


B


C

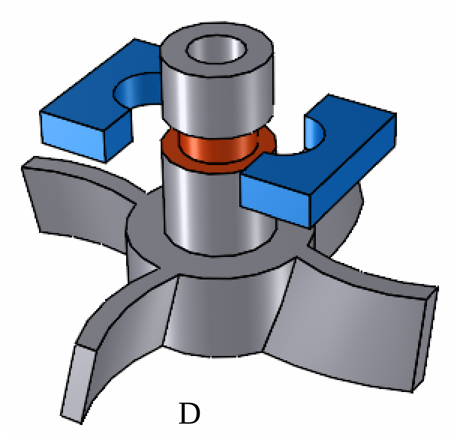

D

Figure 5.7: Side action solids for 4 different test parts (A, B, C, D) shown in retracted state

Another novel aspect of this work that has been illustrated in Figure 5.8 is the customizability of the molding cost function. By changing the parameters, one can get altogether different designs. For example, if the value of γ is decreased by a factor of 10 and the value of $C^{/}$ is increased by 10 times from the previous setting, 2 and 3 side action solids are obtained for the two parts instead of 4 and 4 respectively. In other words, if a higher cost is assigned to assembling and actuating side actions and less cost to manufacturing complex profiles, then lesser number of more complex-shaped side actions are obtained. This feature of the current system has many potential benefits. Depending upon the specifications of the injection molding operation, shop floor personnel simply need to input appropriate parameter values to come up with designs which will minimize the total operational cost for that particular plant or factory.

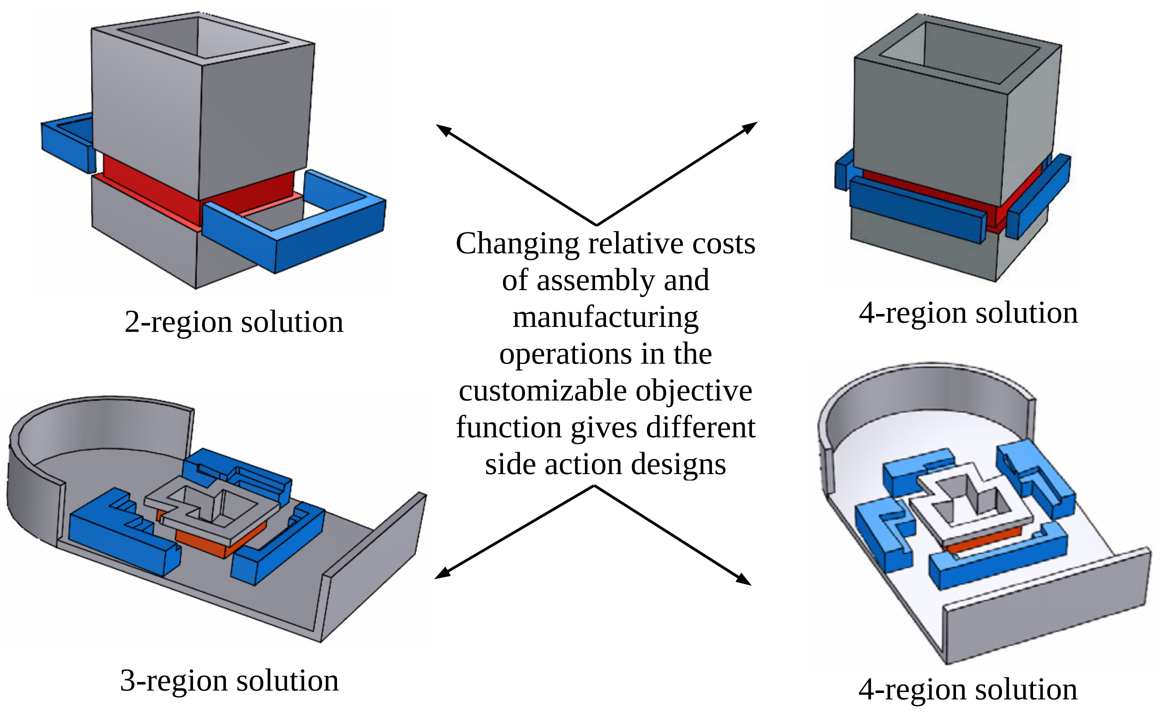

2-region solution
Changing relative costs of assembly and manufacturing operations in the customizable objective function gives different side action designs
4-region solution
3-region solution
4-region solution

Figure 5.8: Alternative designs obtained by modifying the cost function parameters

Chapter 6

CONCLUSIONS

This chapter has been arranged in the following manner. Section 6.1 summarizes the salient features of this research work. Section 6.2 presents the contributions of this work in greater details. Section 6.3 describes the benefits of the work presented in this thesis. Finally Section 6.4 describes the possible future work in this field.

6.1 Summary

In a nutshell, this thesis describes algorithms to generate side action solids to minimize a customizable molding function. Some of the salient features of this approach include designing side actions for undercut regions that are finitely accessible, detecting non-moldable undercuts so that parts can be redesigned immediately, partitioning connected undercut regions into sub-regions that are separately moldable, estimating time complexities of different individual algorithms and showing acceptable performance on typical industrial parts. Although each of these features has been elaborately explained in the subsequent Section, few points need to be noted here. Firstly, the current work overcomes the three problems listed in Section 3.7 which have not been adequately addressed by the existing methods. Secondly, the pre-optimization (geometric) steps of the overall algorithm have a linear, linear-logarithmic or quadratic worst case asymptotic time complexity. This satisfies the desired characteristics of typical computational geometry based algorithms. Thirdly, output (side action set) is generated in a reasonable amount of time (few seconds) for parts whose undercut regions can be molded by 3 or fewer no. of side actions.

However, there are certain shortcomings of this work as well. One of the main limitations is that only two translational motions (with the first one being perpendicular to the main mold opening directions) are considered in this thesis. Although some side actions such as lifters require 3 translational motions, very few industrial parts actually use such retraction mechanisms. Another prominent limitation is that the depth first branch and bound will have an exponential worst-case time complexity. Although this is not a major problem in cases where depth of tree is restricted to 3 or 4, it will definitely increase the running time significantly for undercut regions that need to be sub-divided into 4 or more regions. Alternative optimization strategy needs to be explored for such cases. Finally, another shortcoming of the current work is that uncertainties in cost function parameters have not been dealt with. It is almost impossible to obtain precise data from industrial houses and hence such uncertainties need to be incorporated. Some of these have been explained in details in Section 6.4.

6.2 Contributions

New algorithms to automatically generate shapes of side actions have been presented in this thesis. The major contributions of this work can be summarized as follows.

1. *Designing side actions for finitely accessible undercuts*: As discussed in Chapter 2, existing methods do not address the problem of designing side actions for complex undercuts that are finitely accessible in a satisfactory way. The present work overcomes this serious limitation and successfully designs side actions in such cases.

Fundamentally, the problem with existing approaches was that they considered semi-infinite accessibility in the form of VMaps constructed on Gaussian spheres. Since finite accessibility (feasible retraction spaces) has been computed here, current work does not suffer from this limitation.

2. *Reporting non-moldable undercuts*: Another major issue not adequately addressed in existing methods was the detection of impossible to mold undercuts during the side action generation stage itself, without requiring costly simulation-based post processing operations. The algorithms presented in this thesis correctly reports that certain undercut regions may not have any feasible side action at an intermediate step (See Section 5.2). This ensures that time-consuming optimization step is avoided and the parts can be sent for significant re-design immediately. This is also useful in determining infeasible molding sequences in multi material injection molding as explained in Section 1.3.
3. *Partitioning undercut regions into moldable sub-regions*: Another important problem mentioned in Chapter 2 is the partitioning of connected undercut regions (for which no single side action exists) into smaller regions, so that each sub-region can be molded by a separate side action. The current system accomplishes this task successfully. At the same time, it minimizes a customizable molding cost function. Thus, it serves two purposes simultaneously without any additional overhead.
4. *Estimating time complexities of different algorithms*: Many of the steps in the computation of candidate retraction space and discrete set of candidate retractions have linear or linear-logarithmic worst-case asymptotic time complexities. Few grow quadratically with an increase in the total number of part facets as well the number of

undercut facets. This has been established theoretically and later validated by a series of computational experiments.

5. *Showing acceptable system performance on different parts*: If a connected undercut region can be molded by 3 or fewer number of side actions, then empirical results suggest that this system is capable of finding a solution very close to the optimum in a reasonable amount of time for most practical parts. This is an acceptable performance which will hopefully provide the foundation for developing fully automated software for designing side actions in the near future.

6.3 Benefits

Currently, injection molding of complex, industrial parts is a time-consuming process and is mostly done manually. Although CAD systems like ProEngineer provide tools to automatically design two-piece molds for a given part, often it is unable to design side actions correctly without any form of manual intervention. Consequently, cost and lead-time associated with new injection molded products is high. It is anticipated that the current work will significantly affect the manufacturing sector in the following ways:

1. *Estimating cost during design phase itself*: In the age of concurrent engineering, the present system will go a long way in ensuring its success in the field of injection molding. Traditionally, design and manufacturing used to proceed along separate paths. In what is commonly known as “over the wall” manufacturing, designers and manufacturers rarely used to interact with each other. After finishing a design, designers used to hand it over to the manufacturing people. Usually, a product could never be manufactured defect-free in the first attempt. Manufacturing people would

then hand over this defective design back to the designers for suitable modifications. This cycle used to go on endlessly until both parties were totally satisfied. Naturally, lot of time, money and resources were wasted. So the concept of concurrent engineering was proposed to eliminate such wastage. Now, designers and manufacturers interact at the time of designing a component only to ensure that it can be manufactured easily. Of course, fast, automated design and subsequent testing by performing simulation in Virtual Environment (VE) or by fabricating prototypes by Rapid Prototyping (RP) technologies form an important part of concurrent engineering. However, on top of this, if the actual cost of manufacturing (molding) a part can be estimated with a reasonable degree of accuracy during the design phase itself in order to come up with designs that minimize it, the fruits of concurrent engineering can be realized in its totality. The current system precisely achieves this objective.

2. *Automating mold design to a large extent*: Mold design in an extremely complex operation. It involves design of the mold pieces, the ejector system and the cooling system. The present work only addresses the issue of automatically designing a particular type of mold piece, namely side action. However, this has tremendous impact in a much broader sense. As briefly touched upon in the previous paragraph, automated design of molds is an essential cog in the wheel of concurrent engineering. Some of its manifold benefits are listed as follows:
    a. It allows us to explore various design options and select the best one. In fact, one can iteratively go on improving the design until a completely satisfactory one is obtained.

b. It completely eliminates the cost of manual labor since it only involves one-time installation cost of the relevant software.

c. It reduces the lead time for new plastic products in markets significantly since the multiple month-long cycles of design, fabrication, re-design and re-fabrication are avoided. This enhances the competitiveness of injection molding industries.

6.4 Future Work

Although it is anticipated that the present work will provide the foundations for developing fully automated software for designing side actions in injection molding, further work needs to be done in the following areas to address this problem in its entirety.

1. *Handling side actions other than side cores:* This research work focuses on a particular type of side action, commonly known as side core in molding terminology. Such an emphasis was put deliberately as side cores constitute majority of the side actions used in industry. However, further work needs to be done to generalize our method to design other kinds of side actions, namely, split cores, lifters etc. which may require more than two translational motions for complete disengagement.
2. *Improving edge sub-division scheme:* Work will continue on improving the edge sub-division scheme, so that one can come up with stronger theoretical results to rule out such operations for a majority of candidate retraction space polygonal edges. Essentially, such results will be based on the visibility of the edges from the origin.

Exploration of this idea has already been initiated and plan is to integrate it within the overall framework soon. This is expected to reduce the computation time further.

3. *Handling freeform surfaces without faceting*: Another possible area of research is to extend these ideas to directly design side actions for parts represented by freeform surfaces such as Bezier, NURBS (Non-Uniform Rational B-Spline) etc. Right now, all surfaces (planar or curved) are faceted (triangulated) and then side actions are generated not for the original CAD model, but for its polyhedral version. However, this may not be always acceptable in the industry and hence alternative ways have to be explored to compute the feasible retraction spaces.
4. *Handling side actions where the first translation may not be horizontal*: Theoretically all the algorithms can be extended to deal with cases where the side actions need not be necessarily retracted in a plane perpendicular to the mold opening directions. Instead of transforming the collision polyhedrons and inaccessible voxels into 2D plane, if a 3D collision-free and vertically accessible translation space are computed, then the same idea of carrying out state space search over all non-dominated 3D retractions can be used to solve the problem. This idea needs to be investigated in greater details before it can be implemented in future.
5. *Designing side action actuator mechanisms*: As discussed briefly in Sub-section 1.2.1, different mechanisms are used to actuate side actions. If the side action occupies an entire section of the mold, then a *slide* mechanism is typically used [Bryc98]. The face of the slide is an integral portion of the core or cavity image and it “shuts-off” or creates a sealing area to contain the incoming molten plastic. On the other hand, if the side action only includes a small portion of the mold section, that

activity is performed using a cam. Cams normally do not travel very far and often contain a core pin that would travel in and out of the sidewall in the core or cavity image. They are commonly actuated using either a tapered block or a small angled pin. Another activation system, known as lifter, is sometimes used to form features on the internal portion of the molded part. They are attached to the ejector system so that they not only move back and forth to form the internal feature, but also travel up with the part as the ejector system pushes it out of the mold and back as the ejector system is positioned for the next cycle. Thus, it can be easily concluded that the choice of actuation mechanism will depend upon the retraction length and the type of retraction (core or cavity). Further work needs to be done to automatically select and design such a mechanism based on these parameters.

6. *Handling uncertainties in cost function parameter values*: The current work does not consider the uncertainties in cost function parameter values. However, real world industrial data will always have a significant amount of uncertainty associated with it. Hence, it may be useful to handle them appropriately in future. Different approaches have been suggested in literature to perform so-called robust optimization with respect to uncontrolled variations in environmental and operational parameters. They can be broadly classified into three categories. Stochastic approaches use probability information (mean and variance) of the parameters to minimize the solution sensitivities. Representative work in this field includes Parkinson et al [Park93], Yu and Ishii [Yu98], Jung and Lee [Jung02] for objective function robustness. However, the main shortcoming is that probability distributions need to be known or assumed. Deterministic methods form the second category. Most of them use gradient based

information, including the seminal work by Taguchi [Tagu78], Balling et al. [Ball86], Sundaresan et al. [Sund93], Lee and Park [Lee01] and so on. One of the main limitations of these methods is that the objective functions must be differentiable with respect to the parameters. To address these shortcomings, a non-gradient based method has been developed by Gunawan and Azarm [Guna05] and Li et al. [Li06]. Another class of methods has recently become popular in order to handle epistemic or subjective uncertainties where there is significant ignorance or lack of information about the parameters. Evidence theory [Ober02, Bae04, Mour06] is the most general form of this technique. Possibility theory [Klir98] handles a subset of problems where there is no conflicting evidence in the data. In almost all such cases, fuzzy set theory [Zad65] has been extensively used to characterize and propagate input uncertainties. The best approach for the current work needs to be explored in greater details.

Appendix

A. Regularized and Non-regularized Boolean Operations

Before explaining regularized Boolean operations, few mathematical terms need to be defined formally. Without going into the details of topological spaces, they can be stated simply as follows:

- *Interior*: Given a set $X \subseteq R^3$, $p \in X$ is an interior point of *X*, iff $\in \delta > 0$ such that every point *q* lying with distance δ from *p* also belongs to *X*. Set of all interior points is called the interior of the set; it is denoted by *i*(*X*).
- *Closure*: Closure of *X* is the set of all points *p* such that $\forall \delta > 0$, there is at least one point in *X* within a distance δ of *p*. It is denoted as *k*(*X*).
- *Regular*: A set X is regular if *k*(*i*(*X*)) = *X*.
- *Solid*: A solid is a regular, connected and bounded set with semi-analytical (finite number of infinitely differentiable) boundaries. By bounded set, one refers to a set that can be enclosed by an open ball of finite radius.

Hence, regularized Boolean operations are those Boolean operations such as union, intersection, difference etc. performed on two or more solids such that the above mentioned condition of regular set is satisfied for the output solid (set). Non-regularized Boolean operations do not satisfy this property; however, they are often more intuitive. Hence, both these types of operations are present in any commercial CAD system. However, the regularized operation is carried out by default since it preserves the strict mathematical definition of a solid at every stage. The differences in output after performing these two types of intersection operations on identical solids *A* and *B* are shown in Figure A.1.

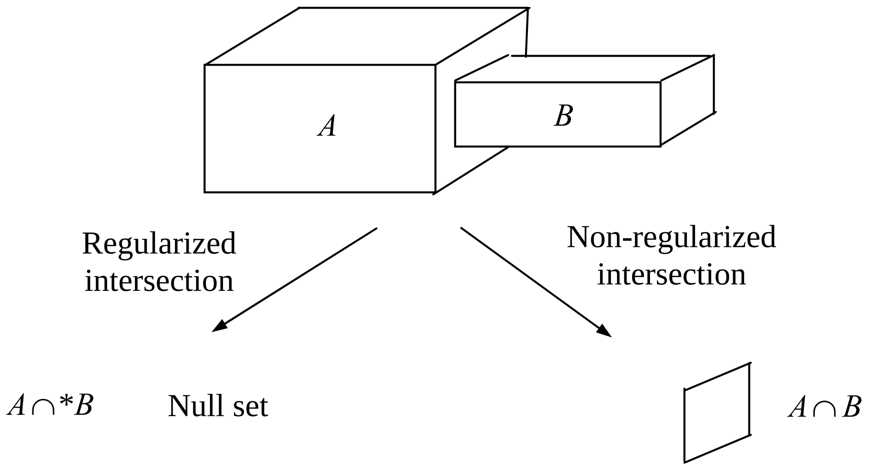


Figure A.1: Differences between regularized and non-regularized Boolean operations

B. Worst-case Asymptotic Time Complexity (O-notation)

The order of growth of the running time of an algorithm gives a simple characterization of the algorithm's efficiency and also allows one to compare the relative performance of alternative algorithms [Corm01]. Although sometimes the exact running time of an algorithm can be obtained, it is sufficient to consider the worst-case running time. This is because the multiplicative constants and lower-order terms of an exact running time are dominated by the effects of the input size itself for large enough inputs.

In other words, when one looks at input sizes large enough to make only the order of growth of the running time relevant, one is studying the *asymptotic* efficiency of algorithms. That is, one is concerned with how the running time of an algorithm increases with the size of the input in the limit, as the size of input increases without bound.

Usually, an algorithm that is asymptotically more efficient will be the best choice for all but small inputs.

O-notation is used to denote only an asymptotically upper bound. For a given function $g(n)$, O($g(n)$) denotes the set of functions given by O($g(n)$) = {$f(n)$: there exist positive constants c and n_o such that $0 \le f\left(n\right) \le cg\left(n\right) \forall n \ge n_0$ }. Using O-notation, the running time of an algorithm can often be described merely by inspecting its overall structure. For example, if any doubly nested for loop structure is present in an algorithm, it may be immediately concluded that there is an O(n^2) upper bound on the worst-case asymptotic running time. The analysis is very simple. The cost of each iteration in the inner loop is bounded from above by O(1) (constant), the two loop indices, say i and j are both at most n, and the inner loop is executed at most once for each of the n^2 pairs of values for i and j. Another important point to be noted here is that when O-notation is used to bound the worst-case running time of an algorithm, a bound is obtained on the running time of the algorithm for every input.

BIBLIOGRAPHY

[Ahn02] H.K. Ahn, M. Berg, P. Bose, S.W Cheng, D. Halperin, J. Matoušek, and O. Schwarzkopf. Separating an Object from Its Cast. *Computer Aided Design*, Vol. 34, No. 8, pp. 547-559, 2002.

[Aron97] B. Aronov, and M. Sharir. On Translational Motion Planning of a Convex Polyhedron in 3-Space. *Siam Journal of Computing*, Vol. 26, No. 6, pp. 1785-1803, 1997.

[Bae04] H. R. Bae, R. V. Gandhi and R. A. Canfield. An Approximation Approach for Uncertainty Quantification Using Evidence Theory. *Reliability Engineering & System Safety*, Vol. 86, pp. 215-225, 2004.

[Ball86] R. J. Balling, J. C. Free and A. R. Parkinson. Consideration of Worst-Case Manufacturing Tolerances in Design Optimization. *ASME Journal of Mechanical Design*, Vol. 108, pp. 438-441.

[Bane06a] A. G. Banerjee, and S. K. Gupta. A Step Towards Automated Design of Side Actions for Injection Molding of Complex Parts. Geometric Modeling and Processing, 2006, *Lecture Notes in Computer Science*, Vol. 4077, pp. 500-513, 2006.

[Bane06b] S. K. Gupta, A. G. Banerjee, X. Li, and G. Fowler. Development of a Manufacturability Analysis Framework for Injection Molded Multi-Material Objects. In *NSF Design, Service, and Manufacturing Grantees and Research Conference*, St. Louis, MO, 2006.

[Boot02] G. Boothroyd, P. Dewhurst, and W. Knight. *Product Design for Manufacture and Assembly,* 2nd Edition. Marcel Dekker, Inc.: New York, New York, 2002.

[Bryc96] D. M Bryce. *Plastic Injection Molding, Vol. I: Manufacturing Process Fundamentals*. Society of Manufacturing Engineers: Dearborn, Michigan, 1996.

[Bryc98] D. M Bryce. *Plastic Injection Molding, Vol. III: …mold design and construction fundamentals*. Society of Manufacturing Engineers: Dearborn, Michigan, 1998.

[Cgal04] Cgal.org. Computational Geometry Algorithms Library. http://www.cgal.org, 2004.

[Corm01] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*, The MIT Press, 2001.

[Chen93] L.L. Chen, S.Y. Chou, and T.C. Woo. Parting Directions for Mould and Die Design. *Computer Aided Design*, Vol. 25, No. 12, pp. 762-768, 1993.

[Chen95] L.L. Chen, S. Y. Chou, and T. C. Woo. Partial Visibility for Selecting a Parting Direction in Mould and Die Design. *Journal of Manufacturing Systems*, Vol. 14, No. 5, pp. 319-330, 1995.

[Chen97] Y. H. Chen. Determining parting direction based on minimum bounding box and fuzzy logics. *International Journal of Machine Tools and Manufacture*, Vol. 37, No. 9, pp. 1187-1199, 1997.

[Chen02] Y. Chen and D.W. Rosen. A Region Based Method to Automated Design of Multi-Piece Molds with Application to Rapid Tooling, *Journal of Computing and Information Science in Engineering*, Vol. 2, No. 2, pp. 86-97, 2002.

[Chen03] Y. Chen and D.W. Rosen. A reverse glue approach to automated construction of multi-piece molds, *Journal of Computing and Information Science in Engineering*, Vol. 3, No. 3, pp. 219-230, 2003.

[Chen06] X. Chen and S. McMains. Finding All Undercut-free Parting Directions for Extrusions. Geometric Modeling and Processing, 2006, *Lecture Notes in Computer Science*, Vol. 4077, pp. 514-527, 2006.

[deBe00] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational geometry: algorithms and applications*, *Second Edition*. Springer-Verlag, Berlin, 2000.

[Dhal01] S. Dhaliwal, S.K. Gupta, J. Huang, and M. Kumar. A feature based approach to automated design of multi-piece sacrificial molds. *ASME Journal of Computing and Information Science in Engineering*, Vol. 1, No. 3, pp. 225--234, 2001.

[Dhal03] S. Dhaliwal, S.K. Gupta, J. Huang, and A. Priyadarshi. Algorithms for Computing Global Accessibility Cones. *Journal of Computing and Information Science in Engineering*, Vol. 3, No. 3, pp. 200-209, 2003.

[Elbe05] G. Elber, X. Chen, and E. Cohen. Mold Accessibility via Gauss Map Analysis. *Journal of Computing and Information Science in Engineering*, Vol. 5, No. 2, pp. 79-85, 2005.

[Faga00] A. Fagade, and D. O. Kazmer. Early Cost Estimation for Injection Molded Components. *Journal of Injection Molding Technology*, Vol. 4, No. 3, pp.97-106, 2000.

[Fowl04] G. Fowler. *Cost and Performance Evaluation Models For Comparing Traditional and Multi-Shot Injection Molding*. Master of Science Thesis, University of Maryland, College Park, 2004.

[Fu99a] M. W. Fu, J. Y. H. Fuh, and A. Y. C. Nee. Generation of optimal parting direction based on undercut features in injection molded parts. *IIE Transactions* Vol. 31, pp. 947-955, 1999.

[Fu99b] M.W. Fu, J.Y.H. Fuh, and A.Y.C. Nee, Undercut Feature Recognition in an Injection Mould Design System. *Computer Aided Design*, Vol. 31, No. 12, pp. 777-790, 1999.

[Gant93] M. A. Ganter, and P. A. Skoglund. Feature extraction for casting core development. *ASME Journal of Mechanical Design*, Vol. 115, No. 4, pp. 744-750, 1993.

[Guna05] S. Gunawan and S. Azarm. Multi-Objective Robust Optimization Using a Sensitivity Region Concept. *Structural and Multidisciplinary Optimization*, Vol. 29, No. 1, pp. 50 – 60, 2005.

[Huan03] J. Huang, S. K. Gupta, and K. Stoppel. Generating sacrificial multi-piece molds using accessibility driven spatial partitioning. *Computer Aided Design*, Vol. 35 No. 3, pp. 1147-1160, 2003.

[Hui92] K. C. Hui, and S. T. Tan. Mould design with sweep operations-a heuristic search approach. *Computer Aided Design*, Vol. 24, No. 2, pp. 81-91, 1992.

[Hui97] K C Hui. Geometric Aspects of the Mouldability of Parts. *Computer Aided Design*, Vol. 29, No. 3, pp. 197-208, 1997.

[Jung02] D. H. Jung and B. C. Lee. Development of a Simple and Efficient Method for Robust Optimization. *International Journal of Numerical Methods in Engineering*, Vol. 23, pp. 2201-2215, 2002.

[Khar06] R. Kharderkar, G. Burton, and S. McMains. Finding feasible mold parting directions using graphics hardware. *Computer Aided Design*, Vol. 38, No. 4, pp. 327-341, 2006.

[Klir98] G. L. Klir and T. A. Filger. *Fuzzy Sets, Uncertainty, and Information*. Prentice-Hall: Englewood Cliffs, New Jersey, 1998.

[Kuma02] M. Kumar and S.K. Gupta. Automated design of multi-stage molds for manufacturing multi-material objects. *ASME Journal of Mechanical Design*, Vol. 124, No. 3, pp. 399-407, 2002.

[Lee01] K. Lee and G. Park. Robust Optimization Considering Tolerances of Design Variables. *Computers and Structures*, Vol. 79, pp. 77-86.

[Li04] X. Li and S.K. Gupta. Geometric Algorithms for Automated Design of Rotary-Platen Multi-Shot Molds. *Computer Aided Design*, Vol. 36, No. 12, pp. 1171-1187, 2004.

[Li06] M. Li, S. Azarm and A. Boyars. A New Deterministic Approach Using Sensitivity Region Measures for Multi-Objective Robust and Feasibility Robust Design Optimization. ASME Journal of Mechanical Design, Vol. 128, pp. 874-883, 2006.

[Liou90] W. T. Liou, J. J-M. Tan, and R. C. T. Lee. Minimum Rectangular Partition Problem for Simple Rectilinear Polygons. *IEEE Transactions on Computer-Aided Design*, Vol. 9, No. 7, pp. 720-733, 1990.

[Lu00] H.Y. Lu and W.B. Lee. Detection of Interference Elements and Release Directions in Die-cast and Injection-moulded Components. *Proceedings of the Institution of Mechanical Engineers*, Vol. 214, No. B6, pp. 431-441, 2000.

[McMa06] S. McMains and X. Chen. Finding Undercut-Free Parting Directions for Polygons with Curved Edges. *Journal of Computing and Information Science in Engineering*, Vol. 6, No. 1, pp. 60-68, 2006.

[Mour06] Z. P. Mourelatos and J. Zhou. A Design Optimization Method Using Evidence Theory. *ASME Journal of Mechanical Design*, Vol. 128, pp. 901-908, 2006.

[Ober02] W. L. Oberkampf and J. C. Helton. Investigation of Evidence Theory for Engineering Applications. *AIAA Non-Deterministic Approaches Forum*, No. AIAA 2002-1569, Denver, Colorado, 2002.

[Park93] A. Parkinson, C. Sorensen and N. A. Pourhassan. General Approach to Robust Optimal Design. *ASME Journal of Mechanical Design*, Vol. 115, pp. 74 – 80, 1993.

[Poli01] C. Poli. *Design for Manufacturing, A Structured Approach*. Butterworth-Heinemann: Boston, Massachusetts, 2001.

[Priy02] A. K. Priyadarshi. *Geometric Algorithms for Automated Design of Multi-Piece Permanent Molds*. Master of Science Thesis, University of Maryland, College Park, 2002.

[Priy04] A.K. Priyadarshi and S.K. Gupta. Geometric algorithms for automated design of multi-piece permanent molds. *Computer Aided Design*, Vol. 36, No. 3, pp. 241-260, 2004.

[Priy06] A. K. Priyadarshi, and S. K. Gupta. Finding Mold-Piece Regions Using Computer Graphics Hardware. Geometric Modeling and Processing, 2006, *Lecture Notes in Computer Science*, Vol. 4077, pp. 655-662, 2006.

[Rapp94] D. Rappaport, and A. Rosenbloom. Moldable and castable polygons. *Computational Geometry: Theory and Applications*, Vol. 4, No. 4, pp. 219-233, 1994.

[Ravi90] B. Ravi, and M. N. Srinivasan. Decision criteria for computer-aided parting surface design. *Computer Aided Design*, Vol. 22, No. 1, pp. 11-18, 1990.

[Rosa00] D. V. Rosato, D. V. Rosato, and M. G. Rosato. *Injection Molding Handbook, Third Edition*. Kluwer Academic Publishers: Boston, Massachusetts, 2000.

[Shin93] K. H. Shin, and K. Lee. Design of Side Cores of Injection Mold from Automatic Detection of Interference Faces. *Journal of Design and Manufacturing*, Vol. 3, No. 4, pp. 225-236, 1993.

[Sund93] S. Sundaresan, K. Ishii and D. R. Houser. A Robust Optimization Procedure with Variation on Design Variables and Constraints. *Advances in Design Automation*, Vol. 65, No. 1, pp. 379-386.

[Tagu78] G. Taguchi. Performance Analysis Design. *International Journal of Production Research*, Vol. 16, pp. 521-530, 1978.

[Wein97] M. Weinstein, and S. Manoocheri. Optimum Parting Line Design of Molded and Cast Parts for Manufacturability. *Journal of Manufacturing Systems*, Vol. 16, No. 1, pp. 1-12, 1997.

[Wong98] T. Wong, S.T. Tan, and W.S. Sze. Parting line formation by slicing a 3D CAD model. *Engineering with Computers*, Vol. 14, No. 4, pp. 330-343, 1998.

[Wuer97a] D. Wuerger, and R. Gadh. Virtual Prototyping of Die Design Part One: Theory and Formulation. *Concurrent Engineering: Research and Applications*, Vol. 5, No. 4, pp. 307-315, 1997.

[Wuer97b] D. Wuerger, and R. Gadh. Virtual Prototyping of Die Design Part Two: Algorithmic, Computational and Practical Considerations. *Concurrent Engineering: Research and Applications*, Vol. 5, No. 4, pp. 317-326, 1997.

[Ye01] X. G. Ye, J. Y. H. Fuh, and K. S. Lee. A hybrid method for recognition of undercut features from moulded parts. *Computer Aided Design*, Vol. 33, No. 14, pp. 1023-1034, 2001.

[Ye04] X. G. Ye, J. Y. H. Fuh, and K. S. Lee. Automatic Undercut Feature Recognition for Side Core Design of Injection Molds. *Journal of Mechanical Design*, Vol. 126, pp. 519-526, 2004.

[Yin01] Z. Yin, H. Ding, and Y. Xiong. Virtual Prototyping of Mold Design: Geometric Mouldability Analysis for Near-net-shape Manufactured Parts by Feature Recognition and Geometric Reasoning. *Computer Aided Design*, Vol. 33, No. 2, pp. 137-154, 2001.

[Yu98] J. C. Yu and K. Ishii. Design for Robustness Based on Manufacturing Variation Patterns. *ASME Journal of Mechanical Design*, Vol. 120, pp. 196-202, 1998.

[Zade65] L. A. Zadeh. *Fuzzy Sets*. Information and Control, Vol. 8, pp. 338-353, 1965.