Műveleti sorrendtervezés az autóipari lézerhegesztés területén

TDK Dolgozat

Szerző:
Buzga Viktor

Konzulensek:
Dr. Kovács András
Dr. Strausz György

2012. október 24.

Tartalomjegyzék

1. Bevezetés

A technológia fejlődése egy öngerjesztő folyamat, melyben az egyes módszerek által kiváltott változások új lehetőségeket nyitnak más területeknek a megújulásra. Napjainkban azok az ipari vállalatok képesek fennmaradni a piacon, amelyek jobban tudnak alkalmazkodni a változások adta lehetőségekhez, könnyebben igazodnak a megjelenő újdonságokhoz. Lényeges pontja az átalakulásnak, hogy az új technológiában milyen alternatív optimalizálási lehetőségek rejlenek, és ezeket milyen mértékig tudja egy vállalat kiaknázni, annak érdekében, hogy ezzel növelje a profitját.

Az említett tendencia alól nem kivétel az autóipar sem, az itt érdekelt szereplőknek is követniük kell a tudomány fejlődését, és adott esetben változtatniuk kell a technológián. Egy ilyen átalakulás figyelhető meg a hegesztési technológiák területén is, ahol manapság egyre jobban terjed és mostanra széles körben használt a távoli lézerhegesztés (Remote Laser Welding, RLW).

A technológia lényege, hogy a korábbi hegesztési eljárásokkal szemben a robotnak nem kell megérintenie vagy megközelítenie a munkadarabot, hanem a munkát a felülettől távol, mintegy 0.5-1.5 méteres távolságból végzi egy lézernyaláb segítségével. A robotfej mozgása lassú, azonban a fejben lévő tükrök segítségével egyetlen pontból nagy felületet képes elérni, így gyorsabb lehet elődeinél.

Több nagy vállalat, mint a BMW, a Volkswagen és az AUDI [9], használja a későbbiekben ismertetett technológiát, mely minőségében, és sebességében is jobb eredményekkel rendelkezik a korábban használt módszerekkel szemben.

Az új eljárás új problémákat vet fel, amelyek megoldásra várnak annak érdekében, hogy hatékonyan tudják alkalmazni, és a költségeket csökkenteni lehessen. A problémák közül kiemelkedik a varratok hegesztési sorrendjének meghatározása, valamint a robotfej pályájának kiszámítása.

Célunk egy olyan algoritmus elkészítése, és bemutatása volt, amely képes kiszámítani egy közel optimális varratsorrendet, valamint a robot mozgását a munkadarab körüli térben. Az alkalmazás elkészítésekor figyelembe vettük azokat a feltételeket, melyeket be kellett tartanunk annak érdekében, hogy a hegesztés megfelelő minőségű legyen. Ezekről a feltételekről szót ejtünk majd a későbbiekben, amikor a technológia sajátosságait tárgyaljuk.

Az elkészült algoritmus által adott eredményeket követve a robot képes lesz a technológia adta lehetőségeken belül az összes varrat jó minőségű hegesztésére a lehető legkisebb műveleti idővel. Egy ilyen rendszer használatával az autógyártásban alkalmazott távoli lézerhegesztés technológiája optimalizálható.

2. Technológiai háttér

2.1. A technológia

A terjedő technológia, az RLW, több elemből épül fel. Mindenekelőtt szükségünk van egy berendezésre, mely jó minőségű hegesztés elkészítésére alkalmas lézersugarat állít elő. A sugarat a munkadarab megfelelő részére kell irányítani. Ezt a feladatot a robot végzi, ami a hegesztőfej valamint a fejben lévő tükrök mozgatásával a megfelelő irányba és pozícióba állítja a nyalábot. A tükrök segítségével a robot mozgatása nélkül is képesek vagyunk nagy terület elérésére. A robot kétféle mozgatási képességének sebességei eltérnek egymástól. Míg a tükrök mozgatása rendkívül gyors, addig a csuklók állítása viszonylag lassú folyamat. Így a műveleti idő csökkentése érdekében a tükrök mozgatásából adódó lézerirányítást kell előnyben részesíteni a robotfej pozicionálásával szemben. Mindezek mellett a rendszerben szükség van egy munkadarabra, amelynek a hegesztését el szeretnénk végezni. A munkadarabon varratok helyezkednek el, amelyek egy-egy hegesztési pontot jelölnek. Ezeket a varratokat kell a lézersugárral támadni, hogy az összeillesztést el tudjuk végezni. Egy ilyen rendszer látható az 1. ábrán működés közben.

1. ábra. Távoli lézerhegesztés, hegesztő robot [1]

Amint azt már említettük, a technológia lényege, hogy a robot nem a felülethez közel, hanem attól nagy távolságban, 0.5-1.5 m-re, helyezkedik el

és innen lövi az elemre a lézersugarat. A robot tipikusan 4 kinetikai csuklóval rendelkezik, amelyek széles mozgásteret tesznek lehetővé. Emiatt a munkadarabot körbe tudja járni, és olyan varratok hegesztése is megoldható, amelyeket a régebbi technikákkal nem lehetett elkészíteni. A robot munkáját a technológia is gyorsítja, a korábbi módszerekkel ellentétben a mozgatási és a hegesztési idő átlapolódhat, így elenyésző mértékű lesz az üresjárat [13].

A technológia tanulmányozása során több jellemző, és ezek értéke is felhasználásra került a különböző forrásokból, annak érdekében, hogy az algoritmus számára biztosítsuk a valós tesztkörnyezetet, ahol az iparban is használatos paraméterekkel dolgozhat.

A munkadarabon felvett varratok a tér bizonyos pontjaiból hegeszthetőek csupán. Egy varrat munkálhatóságát a tér egy pontjából két paraméter határozza meg, az egyik a lézer fókusztávolsága illetve a használható maximális beesési szög. A fókusztávolság mértékére nagyon különféle intervallumokat találtunk a szakirodalmakban és promóciós anyagokban ([9], [13], [15], [12]), 250mm-től 1600mm-ig terjed. A másik paraméter, a maximális beesési szög, a munkadarab anyagától és a lézer teljesítményétől függ. A fejezet végén leírtakban megadott technológia sajátossága, hogy bizonyos beesési szög felett a fényvisszaverődésből származó energia aránya olyan nagy, hogy az ebből adódó veszteséget figyelembe véve, a robot már nem képes kellőképpen megolvasztani az anyagot ahhoz, hogy jó minőségű legyen a hegesztés. Ennek a szögnek az értékéről elenyésző forrásadat állt rendelkezésünkre, hiszen a hegesztett anyag összetétele nagyon sokféle lehet. A [15]-ban megadott értéket használtuk kezdeti referenciaként, ott 40°-nak van megadva a beesési szög maximuma.

Ez a két paraméter együttesen egy csonka kúpot határoz meg a térben, melynek magassága a megadott távolságintervallum hossza, míg nyílásszöge a beesési szög duplája. A csonka kúp által kijelölt térrészen belüli minden pontból hegeszthető az adott varrat, ha a láthatóságot semmi nem akadályozza.

Fontos paraméter még a robot mozgási és hegesztési sebessége. Ez előbbi a robothoz tartozik, a kinetikus csuklók mozgatási képességeiből adódik. Emellett lényeges, hogy a varratokat legalább megadott ideig kell hegeszteni annak érdekében, hogy az ismertetésre kerülő módszerrel megfelelő minőségű hegesztéseket készíthessünk. Ez egy maximális sebességet definiál, amivel a lézersugár haladhat a varraton hegesztés közben.

Ezek a paraméterek az algoritmusunk fontos elemei. Bemenő paraméterként adhatjuk meg az értéküket, ezért az algoritmus sok különböző bemenetre – megfelelően definiált munkadarabra – alkalmazható lett. A valós ipari termelési folyamat több olyan tulajdonsággal is rendelkezik, amelyeket az algoritmusban nem használtunk ki. Ennek több oka is lehet, egyrészt néhány

tulajdonság megtalálható valamely más paraméterben, amelyet eredményesebben tudtunk alkalmazni, vagy pedig a tényező nem volt lényeges az algoritmus szempontjából. Ilyen elem például a lézernél használt hullámhossz, amelynek értéke a varratsorrend szempontjából irreleváns, bár befolyásolja a varrathoz szükséges minimális hegesztési időt, azonban az előzetesen kiszámítható, így ez nem ad plusz információt a programhoz. Az 1. táblázatban az egyes paramétereket és az [9], [13], [15], [12], [10] forrásokban adott értéktartományaikat láthatjuk.

Paraméter neve	**Értéktartomány**
Lézer hullámhossza	1030-1085 nm
Szabadságfokok száma	7
Robotfej gyorsulása	8G
Hegesztési távolság (fókusztávolság)	25-160 cm
Maximális beesési szög	6-40°
Hegesztett panelek távolsága	0.1-0.3 mm
Robot mozgási sebessége	200 m/min
Lézerpozicionálás	1000 m/min

1. táblázat. A távoli lézerhegesztéshez kapcsolódó fontos paraméterek.

A lézertechnológiákat két csoportba sorolhatjuk a [9] alapján. Az egyik a hővezetéses hegesztés (heat conduction), a másik a mély behatolásos (deep penetration) módszer. Előbbiben az anyag az elnyelt fényenergiától, annak vezetése közben olvad meg. Ez esetben nagy hegesztési sebességről, de viszonylag kis mélységű behatolásról beszélünk. A másik esetben a behatolási mélység nagyobb. Ehhez jóval nagyobb energiára van szükségünk, és a hegesztés során gőz is felszabadul a hirtelen megolvasztott anyagból. Ez a gőz betömi a behatolás során keletkezett lyukat és lecsapódik. Ahogy a robot elhalad a varrat felett a lyukban lévő olvadt fém végigfolyik a mélyedésen és teljesen egyedi struktúrát hoz létre [14]. Az utóbbi hegesztési formát kulcslyuk hegesztésnek is hívják nagy behatolási mélysége miatt. A forrásban említett oldalon bővebb leírást találhatunk a technológia sajátosságairól.

2.2. A lézerhegesztés előnyei

Az előzőekben bemutatott technológia több szempontból is jobban teljesít elődeinél. Korábban az ipari alkalmazások nagyobb része ponthegesztési technológiákat használt. Ezek közül is kiemelkedik a még ma is széleskörűen alkalmazott ellenállás ponthegesztés (Resistance Spot Welding, RSW). Ennek lényege, hogy az anyagokat két oldalról egy-egy elektródával fogjuk össze. Az elektródákon átfutó áram felmelegíti az összeillesztésre váró anyagokat a hegesztési pontban. Az elektródák ezen kívül erőt fejtenek ki az anyagokra, ami a megolvasztott helyen összepréseli az elemeket, és azok a lehűlés során egybeolvadnak. A technológiáról bővebben a [6] könyvben olvashatunk.

Az autóiparnak alkalmazkodnia kell a változó felhasználói igényekhez és a piac keltette elvárásokhoz. Erősebb szerkezetet próbálnak meg alkotni annak érdekében, hogy az utasok biztonságát maximalizálják. Ezzel párhuzamosan a fogyasztás csökkentése érdekében könnyebb karosszériák kialakítására törekednek [9]. Az ilyen kettős problémák általában új irányvonalak elindítását, más anyagok, technológiák alkalmazását követelik meg. Az előbb említett forrás alapján az RLW képes arra, hogy nagy precizitása és a hegesztési átlapolódások kiszűrése révén könnyebb szerkezetet hozzon létre. Ezen kívül a technológia sajátosságai alapján olyan módszerrel van dolgunk, amely nagyfokú szabadságot biztosít a karosszériaelemek tervezése során, így lehetővé téve, hogy az autó erősebb és kevésbé deformálható legyen. Továbbá a korábban leírt üresjárati mozgás eliminálásával, és az egy pontból elérhető széles hegesztési tartománnyal azt is garantálja, hogy a gyártási folyamat ezen részének munkaideje drasztikusan csökkenni fog, és a nagyobb kezdeti beruházás ellenére az RLW technológia alkalmazása gazdaságilag is kifizetődő.

3. Matematikai modell

A technológia birtokában szükségünk volt egy olyan matematikai modell felépítésére, amelyben a problémánk megoldható. Egy ilyen modell konstruálásához figyelembe kellett vennünk a tudomány által adott korlátokat, döntéseket kellett hoznunk annak érdekében, hogy modellünk kezelhető nagyságú legyen. Mindemellett azonban képesnek is kellett lennie a rendszer megfelelő szintű szimulálására, az ipari használhatóság biztosításához. A következőkben sorra vesszünk a különböző tulajdonságokat, és magyarázatot adunk azok használatának miként jeire.

3.1. Varrat

Varratnak nevezünk egy olyan ponthalmazt a hegesztési térben (lsd. 3.4), melyeket a robot a lézersugár megszüntetése nélkül képes hegeszteni. Egy ilyen öltés elkészítését tekinthetjük a hegesztés elemi műveletének. A varrat a következő tulajdonságokkal adott a bemeneten:

- Kezdőpont és végpont: A varrat két végpontjának koordinátái a háromdimenziós térben.
- Felületi normális: A munkadarabra (lsd. 3.3) állított merőleges a hegesztési pontban.
- Varrat azonosító: Az elemet egyértelműen azonosító numerikus érték.
- Hegesztési sebesség: Maximális sebesség, amellyel a lézer a varraton végighaladva jó minőségű illesztést készíthet. (Értékéből és a varrat hosszából kiszámítható az a minimális idő, amit a robotnak az adott elem hegesztésével kell töltenie.)

A varratok pontszerűek a hegeszthetőség szempontjából, kezdő és végpozíciójuk is elérhető a robotfej egyazon helyzetéből, bár kiterjedésük van, de annak csak a sebességszámítás szemszögéből van jelentősége. Feltételezhető továbbá, hogy diszjunkt ponthalmazok alkotnak csak varratokat, ami a technológiából adódik (lsd. 2. fejezet). Hegesztést csak sík felületen és egyenes vonalban végzünk, így a felületi normális a varrat teljes hosszában állandó. Megszakítás egy elemi tevékenység során nem következhet be.

A hegesztési út a varratokhoz tartozó speciális pontok váltakozó sorozataként épül fel. Minden egyes hegesztendő elemhez tartozik egy be-, és egy kilépési pont. Belépési pontnak nevezzük a hegesztési tér (lsd. 3.4) azon pozícióját, melyen áthaladva a robot megkezdi a varrat hegesztését. Hasonló elgondolások alapján a kilépési pont a hegesztés befejezéséhez tartozó pozíciót jelenti. A program végén kapott út a be-, és kilépési pontok váltakozásából áll elő, ahol felváltva hegesztés, illetve szimpla mozgatás történik.

3.2. Robot

A hegesztést a robot által mozgatott fej végzi, ennek pályáját határoztuk meg a program által.

Az algoritmust egy ágenses környezetben valósítottuk meg, vagyis egy időben, egy munkadarabot csak egy robot hegeszthet. Ennek oka, hogy a projekt során csak ajtó-komplexitású munkadarabokkal kellett dolgoznunk, és az ilyen feladatokra egyetlen robot használata a megszokott.

A korábbi részekben már említettük, hogy pozicionáló szerkezet több szabadságfokkal is rendelkezik, (kinetikus általában 4-5, a tükrök mozgatásából adódó 2-3) ez számunkra annyit jelentett, hogy a robot a hegesztési tér bármely pontjából képes a felület felé irányítani a sugarat, vagyis a pozíciójából adódóan nem kellett korlátokat megfogalmaznunk a mozgására.

A mintaadatokban látható, hogy az ilyen robotoknak a gyorsulása elérheti akár a 8G-t is, ennek értelmében a modellben azt feltételeztük, hogy a robot végtelen gyorsulással tud mozogni. Bár a sebesség nem állandó a hegesztési út folyamán, de annak változásai ugrásszerűnek tekinthetőek, ezekkel az algoritmusunk során tehát nem kellett számolnunk. Az időkalkuláció közben csupán az egyes útszakaszok hosszát, és a hozzájuk tartozó lehetséges maximális sebességeket kellett alapul vennünk.

A hegesztő fejet mozgató eszköz, hasonlóan a varratokhoz, több bemeneti paraméterrel is rendelkezik:

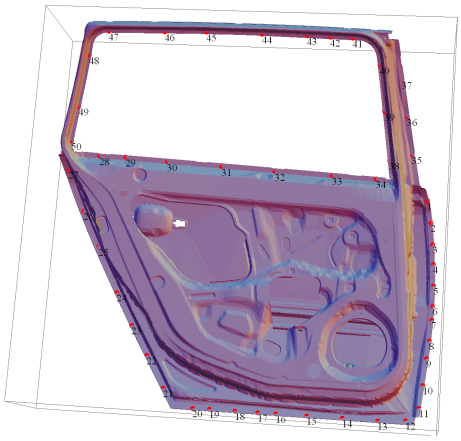
- Maximális sebesség: Az a legnagyobb sebesség, mellyel a robotfej mozogni képes.
- Fókusztávolság intervalluma: A maximális és minimális érték, amelyek közötti fókusztávolság esetén az ebből adódó minőségi kritériumok teljesülnek.
- Beesési szög: A lézersugár és a felületi normális által bezárható maximális szög.

Egy ilyen tulajdonságokkal rendelkező robotot láthatunk a 2. ábrán.

2. ábra. Hegesztő robot [2]

3.3. Munkadarab

Munkadarabnak nevezzük azt a testet, amit hegeszteni szeretnénk. Ezen az elemen varratokat definiáltak számunkra a szakemberek, ahol az alkatrészeket össze kell olvasztanunk. A 3. ábra egy varratpozíciókat is tartalmazó munkadarabot ábrázol.



3. ábra. Munkadarab

A bemenetek és az algoritmus sem tartalmaz olyan elemeket, melyek a valós munkadarab részletes geometriai leírását tárolnák. Mivel a sorrendtervezés során ütközésvizsgálatot nem végeztünk, így a hegesztett elem helyett a rajta megtalálható varratokat használtuk fel a számításaink során, csakis ezekre támaszkodva oldottuk meg a szekvencia kiválasztását.

Munkadarabról tehát nem beszélhetünk, annak csak egyetlen tulajdonságát használtuk fel. Ez a jellemző a tengelyek menti maximális és minimális pozíció. A kapott tulajdonsággal a hegesztési fej által bejárható tér méretét csökkenthettük, így a keresési terünk kiterjedése is kisebb lett. Nincs értelme ugyanis a legkisebb x tengelybeli ponttól a maximális fókusztávolsággal kisebb koordinátánál alacsonyabb x vetületű hegesztési pozíciót vizsgálni, mert onnan nem végezhetünk megfelelő minőségű hegesztést. Ennek értelmében mindhárom koordináta mentén kialakíthatunk egy intervallumot, melyek együttesen egy téglatestet határoznak meg. Ez a testet nevezhetjük hegesztési térnek.

3.4. Hegesztési tér

A feladatot diszkrét térben oldottuk meg. Ennek lényege, hogy az út keresése során a be-, és kilépési pontok csak a 3.3-ban definiált téglatest egy speciális ponthalmazából vehetnek fel értékeket. Az algoritmus futásának kezdetén a térrészben – a bemenetben megadható nagyságú felbontással (rácsállandó) – rácspontokat vettünk fel egyenlő távolságonként. Ezek a pontok a továbbiakban átestek egy címkézésen, amely azt jelenti, hogy minden ponthoz meghatároztuk azokat a varratokat, amelyek hegesztése a technológiai és minőségi korlátok betartása mellett elvégezhetőek a pontból. A korábban említett csonka kúpok átlapolódása miatt egy ponthoz több varrat is tartozhat. Az algoritmusunk további részében csakis ezekkel, a rácspontokkal dolgoztunk, az út keresését ezeken hajtottuk végre.

A megközelítés előnye a folytonos térben reprezentált megvalósítással szemben, hogy nagyon egyszerűen alkalmazhatóak rá olyan megkötések, mint a láthatóság, ütközésmentesség. Láthatóságnak azt nevezzük a hegesztésen belül, hogy egy pontból a varrathoz húzott egyenes szakaszon nincs semmilyen más tárgy, mely lézersugár útját akadályozná. Az elemi láthatóságvizsgálat egy egyszerűbb algoritmus, melyben megbizonyosodhatunk róla, hogy a varrathoz tartozó be- és kilépési pontból az adott varrat az előző definíció alapján látható. Ütközésről ezek mellett akkor beszélhetünk, ha a hegesztési pontban valamilyen más tárgy is tartózkodik (legyen az akár a robotnak egy részegysége, a munkadarab, esetleg valami más, az algoritmusban nem definiált test).

Bár sem ütközésvizsgálatot, sem láthatóságvizsgálatot nem végeztünk a mostani algoritmusban, azok a jövőben egyszerűen implementálhatóak. Ha ugyanis egy hegesztési pontból, bár a távolságok és szögek alapján megfelelő módon hegeszthető egy varrat, nem látunk rá az öltésre, mert egy másik munkadarab, vagy megmunkálandó elemünk egy másik része eltakarja azt, akkor egyszerűen a címkézés során nem kapcsoljuk össze a varratot és a hegesztési pontot. Ezzel megakadályozhatjuk, hogy az adott elem illesztése közben a robot felvehesse ezt a pozíciót mint be-, vagy kilépési pont. Az ütközésvizsgálat is hasonlóan beépíthető a rendszerbe és ezzel a tisztán geometriai számításokat használó, és a hegesztési teret folytonosként kezelő rendszerrel szemben egy nagy előnyre tettünk szert ebben az aspektusban.

Ezen kívül a megközelítés további előnye, hogy a rácspontok pozícióját tetszőlegesen és egyszerűen változtathatjuk. Ezzel bármiféle megkötést és priori tudást vihetünk a rendszerbe csupán a címkézés átalakításával és a lehetséges irányváltási pontok kezelésével. Az előnyeit ennek akkor lehet élvezni, ha valamiféle komplexebb munkadarabbal van dolgunk. Ekkor a struktúrából adódó plusz ismeretek segítségével úgy alakíthatjuk ki a rácspontokat, hogy

azok a lehető legjobban illeszkedjenek az elemhez, és így az algoritmus hatékonyságának növelése, a futási idő csökkentése is lehetséges egyszerre. Ezeket a pozitívumokat a geometriai modell nem rejti magában, így bár jobb megoldásokkal szolgálhat pontosság terén, de adaptivitása és kiterjeszthetősége szűkebb körű.

3.5. A sorrendezési feladat

Az algoritmusunk alapvető célja egy olyan varrat szekvencia és hegesztési útvonal kiszámítása, amely a jelen fejezetben ismertetett modellben egy minimális műveleti idővel rendelkező megoldást definiál. Szekvenciának nevezzünk a bemeneti fájlban adott varratoknak egy bizonyos permutációját, amelynek jelentése, hogy milyen sorrendben haladunk végig a lézerrel az egyes elemeken. A hegesztési útvonalat úgy képzelhetjük el, mint egy háromdimenziós pontsorozatot. Az ebben található pontokon halad keresztül a robotfej olyan módon, hogy az egyes pontok között a két helyvektort összekötő egyenes szakasz mentén halad.

Az algoritmusban használt bemenetek a fejezetben leírtaknak felelnek meg. A hegesztési útvonalhoz tartozik egy bejárási idő, amely alatt a robot az adott háromdimenziós útvonalat be tudja járni a mozgási és hegesztési korlátok betartásával. Ez a hegesztési idő az, amelynek csökkentésével a technológiát használók nagyon sokat nyerhetnek. Azáltal, hogy időt takarítanak meg a sorrend optimalizálásával, növelni tudják profitjukat, energiát spórolnak, több folyamatot tudnak elvégezni adott idő alatt.

Az út pontjait a hegesztési tér pontjaiból választhatjuk ki, a robot csak ezekben a pontokban válthat irányt és kezdheti el, illetve hagyhatja abba a hegesztést. Azokban az időpillanatokban, amikor csak pozicionálást végzünk, vagyis nem történik hegesztés, a lézert kibocsátó fej a lehetséges maximális mozgási sebességével halad.

Hegesztés közben két paramétert kellett alapul vennünk az adott varrathoz tartozó megmunkálási idő kiszámításához. Az egyik a már említett helyváltoztatási sebesség, a másik pedig a varrathoz tartozó minimális hegesztési idő (bemenetként maximális sebesség). Ezek közül azt az időt választottuk az illesztéshez, mint a megmunkálás ideje, amelyik a hosszabb.

A programon belül sem a láthatóságot nem vizsgáltunk, sem ütközésdetektálást nem végeztünk. A hegeszthetőséget ennek értelmében csupán a varratok kezdeti- és végpontja, valamint a középpontjukba állított normálvektor határozta meg.

Nagyon fontos paraméter volt az algoritmus kialakítása során annak futási ideje. Az összes lehetséges hegesztési pontszekvencia permutációt végigvizsgáló algoritmus bár garantáltan a modellben lévő legrövidebb hegesztési

idejű utat találná meg, de futási ideje olyan nagy lenne már kevés varratot tartalmazó esetre is, hogy nem lehetne ipari alkalmazhatóságról beszélni. Gondoljunk csak bele, hogy akár egy 100*100*100-as 10 egységnyi felbontású kockában hányféle szekvencia képzelhető el. Ezek egyenkénti végigszámolása nagyon sok időt emésztene fel, amely meggátolná a program használhatóságát. Célunk volt tehát a hegesztési idő mellett a futási időt is minimalizálni annak érdekében, hogy az algoritmus praktikusságát növeljük. A tesztelés során is figyelemmel kísértük a futási időt és olyan kiegészítő algoritmusokat alkottunk, amelyek a hegesztési idő megtartása mellett képesek a program futási idejét csökkenteni.

4. Algoritmus

Az algoritmusunk az eddigiekben ismertetett modellre (3. fejezet) támaszkodva oldja a 3.5. fejezetben definiált problémát. A megoldás során olyan keresési algoritmusokra támaszkodtunk, melyek hatékonyan tudnak nagy adathalmazzal operálni és képesek a megfelelő úthossz és sorrend felderítésére. A követező fejezet során végighaladunk az algoritmus elméleti vázán, megmutatjuk, hogy milyen módszereket ismertünk meg a forrásokból, amelyek képesek a probléma megoldására, és ezeket hogyan alkalmaztuk, mely elemeiket bővítettük.

Az algoritmus két jól elkülöníthető, jóllehet eltérő komplexitású, feladat megoldásából áll. Először a 3.4 szakaszban ismertetett címkézési feladatot végeztük el, melynek során kialakult a hegesztési tér, majd meghatároztuk a korábban felcímkézett pontokon bejárható, a kritériumokat teljesítő, minimális idő alatt megtehető utat.

4.1. A hegesztési tér címkézése

Az első lépésünk tehát a hegesztési tér előállítása és a pontok kiegészítése azon varratokról szóló információkkal, amelyeket hegeszteni képes. A címkézés fontos része a programnak, hiszen itt dől el, hogy mely illesztéseket milyen térrészből tudjuk megmunkálni. Címkének nevezzük a hegesztési pozícióhoz tartozó listát, melyben azon varratok azonosítói szerepelnek, amelyek megfelelő minőségben hegeszthetőek a pontból.

A munkadarabunk, illetőleg csupán a róla megadott információk – a varratok – kijelölik számunkra azt a térfogatot, amelyben a pontjainkat el kellett helyeznünk. A hegesztési tér határainak kiszámítása több lépésből áll. A var-

ratok legnagyobb x koordinátájú pontjának x pozíciója:

$$MaxXCoordinate = \max\left(\max_i\left(V^x_{Begin}(i)\right), \max_i\left(V^x_{End}(i)\right)\right)$$

ahol $V^x_{Begin}(i)$ az i. varrat belépési pontjának x koordinátája. Hasonló módon kiszámíthatjuk mindhárom irányban a koordináták maximumát és minimumát. Ebből a következő módon kaphatjuk a hegesztési tér határait:

$$WeldingAreaMinX = MinXCoordinate - MaxFocalLength$$
$$WeldingAreaMaxX = MaxXCoordinate + MaxFocalLength$$
$$WeldingAreaMinY = MinYCoordinate - MaxFocalLength$$
$$WeldingAreaMaxY = MaxYCoordinate + MaxFocalLength$$
$$WeldingAreaMinZ = MinZCoordinate - MaxFocalLength$$
$$WeldingAreaMaxZ = MaxZCoordinate + MaxFocalLength$$

ahol a $MaxFocalLength$ a robothoz tartozó maximális fókusztávolság. Ezzel egy elég durva becslését kaptuk a határoknak. Ennél pontosabbat csak akkor tudnánk készíteni, hogyha minden elemhez definiálnánk a hozzá tartozó kúpot, majd ezek köré egy befoglaló testet építenénk, de ez túlságosan sok időbe telne így megelégedtünk egy ilyen szintű becsléssel is.

Ha a tér már rendelkezésünkre állt, akkor felvettük benne a pontjainkat az algoritmus bemeneteként adott távolságban egymástól a három koordináta mentén. A kapott kockarács pontjai közül csak azokat tartottuk meg, amelyekből valamely varrat hegesztése lehetséges, ennek nyilvánvaló oka, hogy így kevesebb ponton kellett a keresést futtatnunk és nem is zártunk ki ezzel a lépéssel olyan pontot, ami a végső út részét képezte volna.

A hegesztési pontnak két kritériumot kell teljesítenie, hogy adott varratot megfelelő módon hegeszthessen:

- $(P - V_{center}) \cdot V_{normal} \leq \cos(MaxIncidenceAngle)$
- $|P - V_{center}| \in (FocalLenghtMin, FocalLenghtMax)$

ahol P a hegesztési tér vizsgált pontja, V_{center} és V_{normal} a varrat középpontját, illetve annak normál vektorát jelöli, a *MaxIncidenceAngle*, *FocalLenghtMin*, és *FocalLenghtMax* pedig a robot bemeneten adott paraméterei (3.2).

Ha ez a két kritérium teljesült, akkor a hegesztési ponthoz tartozó címkébe belekerült a varrat azonosítója, egyébként pedig nem. Ezzel kialakult a hegesztési terünk, illetve a benne lévő pontoknak a halmaza, melyek potenciális jelöltjei a hegesztési útnak. A továbbiakban ezeken az elemeken végeztük az algoritmusban a számításokat.

4.2. A sorrendtervezési algoritmus

Az utazó ügynök feladat (Travelling Salesman Problem, TSP) egy $G = (V, E)$ teljes irányított gráfban a $c : E \to \mathbf{R}_+$ élhosszak esetén a legrövidebb olyan körnek a megkeresése, amely a gráf minden pontján áthalad [11]. A feladatot nagyon sokféle probléma modellezésére használták fel eredményesen, alkalmazhatósága nagyon széles körű. Mivel az optimális megoldás megtalálása NP-teljes feladat [5], így nagy adathalmazra az egzakt algoritmus használata nem célszerű. Ezért különböző metaheurisztikákat használtunk fel, melyek segítségével kezelhető idő alatt juthattunk egy közel-optimális megoldáshoz.

Algoritmus szerkezet

Az algoritmusban a [11] jegyzetben ismertetett kétfázisú módszert követtük. Ebben először egy kezdeti túra építését, majd ennek a javítását végzik el. Túranövelésen a pontoknak a Hamilton-körhöz adását értik, míg javítás során ezeknek a pontoknak egy alternatív szekvenciáját vizsgálják, és ha javulást hozott a változtatás, akkor a módosított túrát tartják meg, míg ha nem, akkor visszatérnek az eredetihez.

A túra növelésére különböző heurisztikák léteznek. Ezek a módszerek a következő túrába illesztendő pontot adják meg. Ha megkaptuk, akkor azt azon két pont közé kell beszúrnunk, amelyek esetén a túra minimális mértékben nő. Ha a következő pont c, és azt az a és b pontok közé illesztjük be a túrába, akkor beszúrás alatt az ab (a-ból a b-be futó) él felbontását, és az ac valamint a cb élek túrába illesztését értjük. Az fentebb említett irodalomban fel vannak sorolva az egyes heurisztikák a pont kiválasztására, illetve ezek használata és hatékonyságuk.

A javító módszerek arra az elvre támaszkodnak, hogy egyszerűen konstruálhatunk új kört egy már meglévőből. Ehhez egy úgynevezett *szomszédság* definiálására van szükségünk. Szomszédságnak általánosságban a lehetséges megoldások kapcsolatát értjük. Két megoldás kapcsolatban van egymással, ha egy meghatározott lépés végrehajtásával egymásba vihetőek. Attól függően, hogy ezt milyen lépésen át tesszük meg különböző típusú szomszédságokat különböztetünk meg.

Egy lehetséges metaheurisztika a túra javításának megoldására a 2-optimális szomszédság használata.

Egy T túrát *2-optimálisnak* (továbbiakban 2-opt) nevezünk, ha tetszőleges uv, ab élpárjára teljesül, hogy

$$c(uv) + c(ab) \geq c(ua) + c(bv)$$

ahol $T - \{uv, ab\} \cup \{ua, bv\}$ szintén túra [11].

A definíció alapján a 2-opt heurisztikára nézve szomszédnak tekintünk két megoldást, hogyha az egyikben lévő uv, ab élpár helyett az ua, vb szakaszokat használva éppen a másikba jutunk. Abban az esetben, hogy ha a változtatás során a körünk két diszjunkt körre esik szét ($T - \{uv, ab\} \cup \{ua, bv\}$ nem túra), akkor az $\{ua, bv\}$ helyett az $\{ub, av\}$-t használva már túrához fogunk jutni. Az algoritmus egy lokális optimumba konvergál, mely egy 2-optimális megoldás.

A TSP megoldásának során ezt a szomszédságot egy hegymászó keresés keretében használtuk. Ilyen típusú lokális keresés esetén megoldás jelöltek hatékonyságát vizsgáltuk meg. Egy ellenőrzésen belül a mindenkori legjobb megoldást viszonyítottuk egy szomszédjához. Ha ez alapján javulást értünk el a hatékonyságban, akkor lecseréltük a túránkat a szomszédra, és onnantól ahhoz viszonyítottunk, annak a szomszédait vettük sorra. Az algoritmus befejeződött abban az esetben, ha egy iterációban már nem volt tovább csökkenthető a túra hossza. Ezzel a módszerrel gyorsabban jutottunk megoldáshoz, hiszen egy szekvencia szomszédsága kezelhető nagyságú.

Az előbb említett megoldásmenethez a források alapján kiválasztottuk a megfelelő részalgoritmusokat és értékeket, amelyekkel megfelelő mértékben alkalmazkodni tudtunk a lézerhegesztés tulajdonságaihoz. A döntések során figyelembe vettük azt, hogy az adott módszer milyen hatékonysággal képes a feladatot megoldani, illetve, ez mennyi időbe telik. A két szempont összehangolása nagy fontossággal bír majd az ipari alkalmazhatóság szemszögéből.

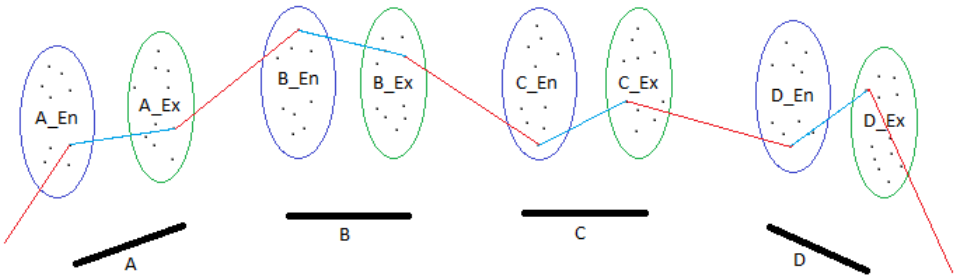
Általánosított TSP

Nagyon sok irodalmat találhatunk ([3], [4]), melyek az általánosított utazó ügynök probléma (Generialized TSP, GTSP) megoldását használják fel a feladataik hatékony teljesítéséhez. A GTSP lényege, hogy a pontok diszjunkt csoportokba vannak osztva és a Hamilton-kör keresése helyett egy olyan kör megtalálása a feladat, amely minden egyes csoportból pontosan 1 elemen halad keresztül.

A sorrendtervezési feladattal egy GTSP-t modelleztünk. Az előző definíció csoportjainak elemeit esetünkben a varratokhoz tartozó hegesztési pontok alkotják. Minden varrathoz két ilyen csoport tartozik, az egyik a belépési, míg a másik a kilépési pontokat tartalmazó ponthalmaz. A hegesztési tér pontjai multiplicitással rendelkeznek, így alkalmazkodtunk ahhoz a tényhez, hogy a be-, és kilépési pontok ugyanazokat a térbeli pozíciókat vehetik fel. Egyszerűen úgy tekintettünk ezekre a pontokra, hogy ugyanannak a pontnak más reprezentációi. Ezeket a címkézésben már ismertetett pontokhoz tartozó listákkal reprezentáltuk a programban. Ennek értelmében a hegesztési útvonal

olyan szakaszokból áll össze, ahol minden csoportból pontosan egy elemet érintünk, ahogy a GTSP definíciójában is szerepel. Fontos kikötés azonban, hogy belépési ponthalmazból, csak a hozzá tartozó varrat kilépési elemeihez vezethet szakasz, és a kilépési pontokból, csak egy másik varrathoz tartozó belépési ponthalmaz valamely elemébe futhat él. Ebben a reprezentációban a multiplicitás kiterjed arra is, hogy vannak olyan hegesztési pozíciók, melyekből több varrat is elérhető. Ilyen esetben a pozícióból nem kettő reprezentánssal van dolgunk, hanem minden egyes varrathoz, mely belőle elérhető, tartozik egy, a többitől független reprezentációjú hegesztési pont.

A 4. ábra a leírt algoritmus szemantikáját mutatja be. A kék halmazok jelölik a varratokhoz tartozó belépési pontokat tartalmazó teret, míg a zöldek a kilépési halmazt. Ez a két ponthalmaz valójában ugyanazoknak a pontoknak különböző reprezentációit tartalmazza, csupán a modellben tekintettük ezeket különbözőnek. A képen látható piros-kék törtvonal egy lehetséges szekvenciát jelöl, amely az $\ldots ABCD \ldots$ sorrendhez tartozik, ahol A, B, C és D varratokat jelölnek.



4. ábra. Hegesztési modell

Visszavezettük tehát a sorrendtervezési feladatunkat egy GTSP-re ezzel a modellel. Ezt a problémát oldottuk meg azoknak az algoritmusoknak az alkalmazásával, melyeket a 4.2. részben ismertettünk, és amelyeket az irodalmakban találhatunk. Fontos megemlíteni, hogy míg mind a TSP, mind a GTSP megoldása során egy kör keresése zajlik, addig nekünk nem volt olyan feltételünk, mely a robot kezdőpozícióba való visszatérését megkövetelné, így esetünkben egy Hamilton-út megtalálása volt a fő cél. Ezt a változtatást az összes részegység implementálása során figyelembe kellett vennünk és olyan módon kellett átalakítani azokat, hogy alkalmazkodjanak a módosított algoritmushoz.

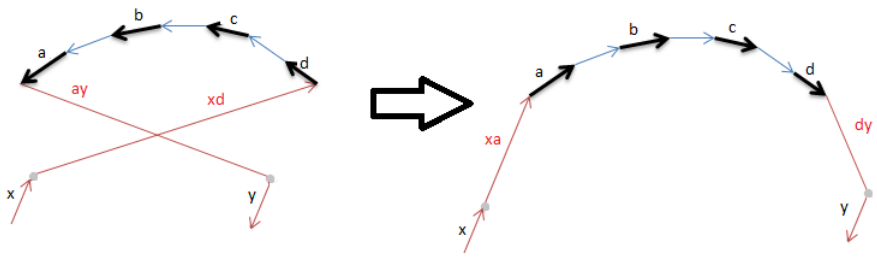
A TSP technikák alkalmazása

A túra felépítéséhez a legtávolabbi pont beszúrásának heurisztikáját választottuk. Ez alapján minden iteráció alkalmával a túrától legtávolabbi pontot illesztettük a szekvenciába minimális költséggel. A választás oka, hogy ez a kiegészítő módszer működik a legjobban a [11] irodalom alapján, Euklideszi távolság használata esetén. Annak értelmében, hogy kör helyett útkeresést végeztünk, ez a lépés is kiegészítésre szorult, hiszen a szélső pontokra is adaptálnunk kellett a technikát, vagyis azokat a lehetőségeket is meg kellett vizsgálnunk, melyeknél a túra szélére szúrjuk be a következő varratot.

A javítás során használt lokális keresési algoritmusban a 2-opt-ot is át kellett alakítanunk minimális mértékben. Bár a varratok sorrendjét változtatjuk, a tényleges csere során a hegesztési pozíciókat is korrigálnunk kellett a GTSP-beli reprezentációnak megfelelően. Ennek értelmében egy varrat sorrendbeli változtatás a következő lépésekből áll:

1. Két, a szekvenciában nem szomszédos (nem egymás utáni) varrat cseréje.
2. A felcserélt illesztések közti varratok szekvencián belüli sorrendjének megfordítása.
3. Az összes, a cserében módosított varrat be-, és kilépési pontjainak felcserélése.
4. Irányváltoztatási pontok korrekciója (lásd később).

A változtatásokat szemlélteti az 5. ábra. a és d a két felcserélt varrathoz, b és c a további módosított varratokhoz tartozó hegesztési szakaszok. A kékkel jelölt nyilak a hegesztések közti üresjárati robotpozicionálásokat jelölik. Ezen kívül míg a bal oldali esetben a szekvencia a $\ldots xdcbay\ldots$, addig a jobb oldaliban $\ldots xabcdy\ldots$.



5. ábra. 2-optimális varratcsere

A sebesség javítása érdekében a túra javítását egy kiegészítő algoritmussal is megtoldottuk. Ennek lényege, hogy a szomszéd megvizsgálása előtt elvégeztünk egy egyszerű elemzést a módosított szekvenciára, és csak abban az esetben vizsgáltuk meg azt, ha teljesít egy kritériumot.

Minden varratpárra meghatároztuk a többi varrattól vett legkisebb távolságát a hegesztési halmazokra nézve, vagyis annak a szakasznak a hosszát, melyet a két varrat hegesztése között a robotnak mindenképpen meg kell tennie. Ekkor ha az előző ábrán látott xd és ay szakaszok összege kisebb mint az x-hez és a-hoz tartozó varratok, valamint a b-hez és y-hoz tartozó illesztések között kiszámított minimális távolságok összege, akkor semmiképpen nem érhetünk el javulást a szomszéd vizsgálatával. Ennek oka, hogy a módosított távolság a legideálisabb esetben is nagyobb, mint a jelenlegi. Időt spórolhatunk meg azzal, ha ezekben az esetekben nem számoljuk ki javulást semmiképpen sem okozó szomszédot.

A minimális távolságra nekünk egy alsó becslés is elég volt, hiszen akkor zárunk ki pontokat, ha ez a hossz a nagyobb. Két hegesztési halmaz közötti legrövidebb távolság meghatározásához ezért egy egyszerű algoritmust találtunk ki. Mindkét halmaz pontjaira meghatároztunk egy-egy befoglaló gömböt, és ezeknek a távolságát vettük a becslés során.

A tesztelés során részletesen elemeztük ennek a lépésnek a hegesztési út teljesítéséhez szükséges időre, illetve az algoritmus futási idejére vett hatását. Ezek az mérőszámok fontos szerepet játszanak abban, hogy a program használható legyen az iparban alkalmazott, sok varratot tartalmazó esetre is.

Az útvonal korrigálása

A növelés és a javítás során is használtunk egy kiegészítő algoritmust, mely az adott varrathoz tartozó be-, és kilépési pontokat állította be a megfelelő helyre a szekvencia többi elemének rögzítése mellett. Ennek oka, hogy egy varratsorrend változtatás során nem garantálható, hogy a korábbi irányváltoztatási pontok továbbra is a legrövidebb utat biztosítják a megelőző és a következő varrat között. Így minden változtatás esetén az illesztéseket meg kellett vizsgálni, és ha volt olyan elemünk, amely korrekcióra szorult, akkor azt javítanunk kellett, hogy mindig a lehető legrövidebb útvonallal számoljunk, és azt használjuk.

Itt is figyelembe vettük a futási időt és próbáltuk azt optimalizálni. Hatásos lépés volt, hogy a korrekció során nem minden lehetséges pontpárt próbáltunk végig, hanem először a belépési pont rögzítésével kerestük a kilépési pont legjobb pozícióját, majd ennek a helynek a rögzítésével optimalizáltuk a kimenetit. Ennek a javításnak is megvizsgáltuk a hatását a tesztelés során.

Ezeket az algoritmusokat és technikákat implementáltuk tehát a progra-

munkba. A következő részben olvashatunk arról részletesebben, hogy ezeket a jól definiált megoldásmódokat hogyan építettük be egy a matematikai modell alapján definiált struktúrába, milyen módszereket használtunk fel, amelyek segítették a szerkezet és a logika kialakítását.

5. Megvalósítás

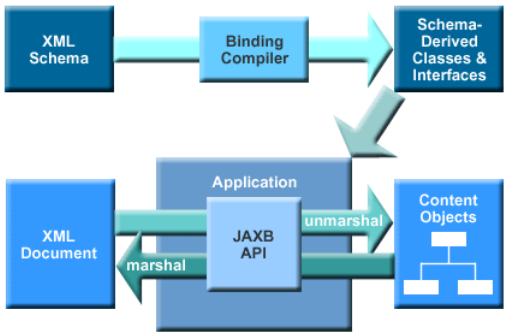
5.1. Eszközök

Az implementáláshoz első lépésben meg kellett választanunk azokat a technológiákat, melyeket felhasználtunk. Olyan programokat, csomagokat kerestünk, melyek képesek garantálni számunkra a szabadságot az alkalmazás felépítése közben, ugyanakkor elég fejlettek ahhoz, hogy egy ilyen volumenű feladat elvégezhető legyen bennük.

Mivel mindenképpen platformfüggetlenséget szerettünk volna biztosítani a felhasználók számára, amit számunkra a legjobban a Java ajánl fel, így ezt választottuk elsődleges programozási nyelvnek. A felépítés során a mindenkori legfrissebb fejlesztői eszköztárral dolgoztunk így a végleges változat a jdk 1.7.0-ás verziókak készült. A fejlesztést az Eclipse SDK 3.7.2-es környezetén belül végeztük el, minthogy ez a platform piacvezető eszköze, ez a választás egyértelmű volt.

Az osztályok létrehozásának egy speciális módját választottuk, hogy a bementek kezelését megkönnyítsük. A kódolás során tulajdonképpen generált java osztályokat egészítettünk ki elemekkel, ruháztunk fel képességekkel. A generálást és a bemenetek kezelését a JAXB (Java Architecture for XML Binding) keretrendszer segítségével valósítottuk meg. Ez a csomag lehetővé teszi számunkra, hogy XML-ben (Extensible Markup Language) definiált bemeneti struktúrákat és rájuk illeszkedő, a bemenő paramétereket tartalmazó fájlokat kezeljünk a programon belül.

A 6. ábrán láthatjuk, hogy hogyan is néz ki keretrendszer felépítése. Ezt a későbbiekben felületesen ismertetjük, főleg az általunk használt elemeket kiemelve. Sokkal részletesebb leírást találunk a kapcsolódó irodalomban, illetve az Oracle weboldalán [8].



6. ábra. JAXB architektúra [7]

Az említett rendszer használatához mindenekelőtt szükségünk volt egy olyan fájlra, amely megmutatja a fordítónak, hogy milyen osztályokat generáljon, és hogyan várja a bemeneteket. A technológia erre az XSD (XML Schema Definition) típust támogatja, melyben XML nyelven íratjuk le, hogy hogyan nézzen ki a bementünk, milyen tartalmú java fájlok jöjjenek létre. Ennek a fájlnak megvan a metodikája, melyet követnünk kellett a megfelelő kezelés érdekében. Több grafikus felület is létezik, melyekkel egyszerűsíthetjük a munka ezen részét. Az Eclipse által támogatott tervezőben állítottuk össze a sémánkat, melynek felépítését a későbbiekben ismertetjük. Több komplex típust is létrehoztunk, melyekből osztályok készültek, és ezeket az osztályokat bővítettük a feladat megoldása érdekében.

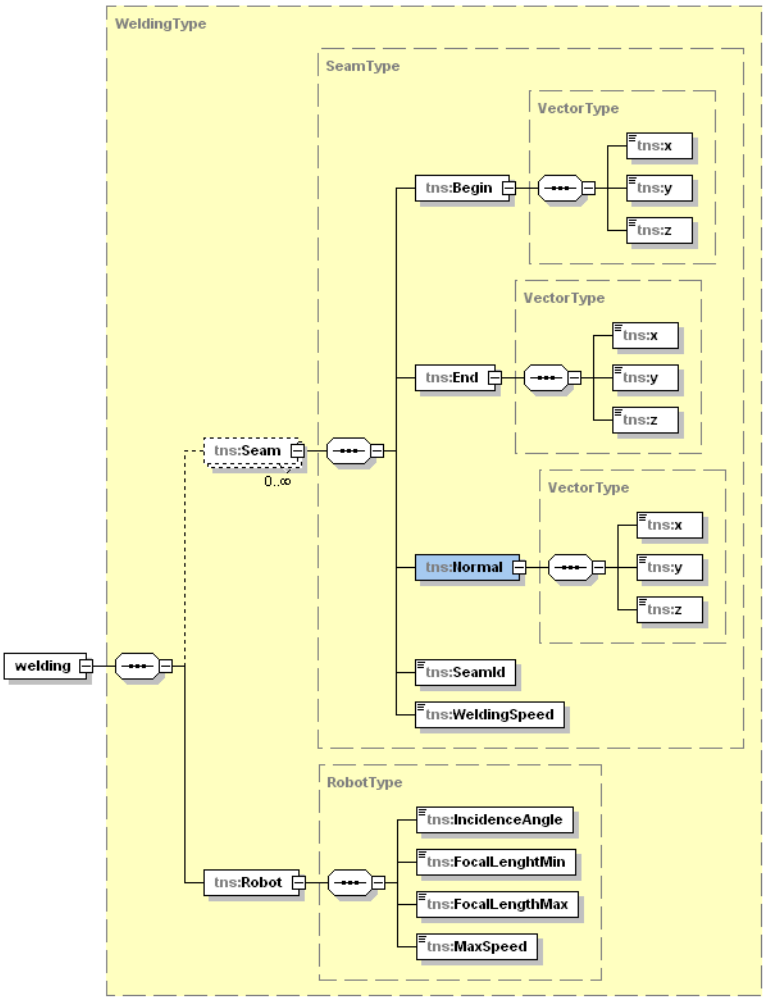
Az XSD-vel a birtokunkban megkezdődhetett a megfelelő osztályok legenerálása. Ehhez segítséget nyújtott az XJC eszköz, mely az említett formátumú fájlokból java osztályokat tud előállítani [7]. A folyamat során több java fájl is létrejött, melyeknek különböző a felhasználásuk. Elsőként figyeljük meg a sémában definiált típusoknak megfelelő osztályokat. Ezekben csupán az inicializálásért felelős és változókezeléssel kapcsolatos függvények szerepelnek. Találhatunk a generált mappában egy csomaginformációt tartalmazó fájlt, valamint 3 a beolvasásért felelős illetve az objektumok megfelelő kezelését biztosító összetett típus, a JAXBUnMarshaller-t, a JAXBMarshaller-t , és az ObjectFactory-t. Ez utóbbi azt biztosítja, hogy a JAXB a típust és a generált osztályt össze tudja kapcsolni, meg tudja találni a megfelelő osztályokat [7]. A másik kettő az XML beolvasásáért, illetve kiírásáért felelős. Mi jelen esetben csak az elsőt használtuk, hiszen nem volt szükségünk egy ilyen struktúrájú kimenetre a programban.

A létrejött osztályokat kiegészítve a megfelelő elemekkel megtörténhetett a fájl beolvasása az UnMarshaller segítségével. Erre egy nagyon egyszerű és gyors módszert biztosít a rendszer. Az XML-eket ugyanis validálhatjuk a

korábban definiált sémára, ami azt jelenti, hogy megvizsgáljuk a felépítést és ha megfelel az XSD-nek, akkor validnak tekintjük, még ellenkező esetben nem. Nem valid bemenetre a program nem fut le, így nem is szolgáltat a definiálás hibájából adódó rossz megoldásokat. Ezzel biztosítható volt, hogy a program csak olyan bemenetekkel dolgozik, amelyek megfelelő formátumban adottak.

5.2. Bementek kezelése, előkészületek

Feladatunk tehát az előző pontban ismertetett elemek elkészítése és összeillesztése volt. A bemenetben definiált elemek a 7. ábrán látható struktúrában adottak.



7. ábra. Bemenetben adott elemek struktúrája

Az előbbi ábra a típusokat tartalmazza, amelyeket generáltunk és ahogy látható, az XML egy *WeldingType* elemet definiál, így ez volt az, amit beolvastunk a JAXB segítségével. Ezek az típusok a technológia egy-egy elemét azonosítják. A *VectorType* egy pontot jelül a munkadarab által definiált térben, a *SeamType* egy varratot határoz meg, míg a *RobotType* a robot specifikációit tartalmazza. Ezeket az adatokat a *WeldingType*-on belülről tudjuk elérni.

Az osztályokat különböző vektoraritmetikai műveletekkel egészítettük ki. Ezek segítettek abban, hogy kiszámítsuk a szükséges távolságokat, szögeket, melyek elengedhetetlenek a feladat megoldásához. A funkciók bővítésével az osztályaink együttesen, egymás elemeit használva képesek voltak az algoritmus vázának felépítésére és így a feladat megoldására.

Ezeken kívül kiegészítő osztályokat is használtunk, melyek a munkadarabot, a hegesztési teret reprezentálták. Ezzel kialakítottuk azt a struktúrát, melynek segítségével az algoritmus implementálásra került és meg tudtuk oldani a sorrendtervezést.

5.3. Hegesztési tér felcímkézése

A program egyik alapköve a hegesztési tér, melyben a varratokhoz tartozó, az illesztési kritériumokat kielégítő háromdimenziós pozíciókat hoztuk létre, illetve címkéztük meg. A jelölés lényege, hogy minden egyes elemhez definiáltunk egy listát, melyben az abból a pontból elérhető varratok azonosítói szerepeltek.

A *WeldingSpace* osztály tartalmazza a címkével kiegészített vektorjegyzéket. Fontos hogy a megjelölés alkalmával csak azokat a pontokat tartottuk meg, melyekhez legalább egy varrat tartozott. Ezzel, mint már ismertettük, csökkenthettük a futási időt, hiszen a program nem töltött felesleges időt olyan számolásokkal, melyek nem adhattak megfelelő eredményt, abból kifolyólag, hogy a hegesztési kritériumok nem teljesültek az általuk kezelt pontokra.

A következő pszeudo-kód mutatja, hogy a címkézés milyen módon történt:

```
FOR EACH x in welding space
    FOR EACH y in welding space
        FOR EACH z in welding space
            FOR EACH seam on welding piece
                incidenceAngle = getAngle()
                IF(incidenceAngle < MaxIncidenceAngle)
                    focalLength = getFocalLength()
                    IF(focalLength BETWEEN
```

```
            MaxFocalLength AND MinFocalLength)
                add seam to (x, y, z)'s label
    IF((x, y, z)'s label is not empty)
        add (x, y, z) to labelledVector
```

Mivel a szög tipikusan kicsi, a távolságintervallum pedig nagy, így időt spóroltunk a fenti sorrend betartásával, hiszen a szög vizsgálata a tér több pontját szűrte ki. A címkézés és pontelhagyás igazából a listába töltést, valamint annak kihagyását jelenti. A funkció lefuttatásával megkaptuk azon pontokat, melyekből elérhető legalább egy varrat.

5.4. Túra növelése

Az utazó ügynök probléma az előbb definiált pontokra épít. A következő kód mutatja, hogy milyen sorrendben történik az elemeknek a túrába illesztése. Látható, hogy három kiemelt lépésünk van, melyek minden iterációt meghatároznak. Beszúrjuk a legtávolabbi varratot, annak legjobb pozíciójába, és korrigáljuk a hegesztési pontokat.

```
LIST weldingSequence = null
FOR EACH seam on welding piece
    nextSeam = seam with maximal distance from weldingSequence
    index = nextSeam's best place in weldingSequence
    INSERT nextSeam TO weldingSequence(index)
    correct entry and exit points of weldingSequence's seams
```

A legtávolabbi varrat, illetve annak pozíciójának a megtalálására egyszerű maximum és minimumkereső algoritmusokat használtunk, a korrekcióra elvégzéséhez a be-, és kilépési pontoknak a külön-külön a többi elem rögzítése mellett elvégzett hegesztési pontokon kereső algoritmust használtuk.

5.5. Túra javítása

A túrát javító algoritmus az elméleti részben ismertetett 2-opt módszert használja, ennek segítségével keres az alternatív megoldások között, hogy megtalálja a legjobb utat, amit a robot követhet. Ezt a lépést mindaddig ismételjük, amíg egy iteráción belül nem változik a szekvencia, hiszen, hogyha változtatunk, akkor a következő iterációban egy másik megoldásjelöltet, illetve annak szomszédságát kell elemeznünk. Ennek értelmében, hogyha nem történik csere, akkor az a további iterációkban sem fog bekövetkezni, így leállíthatjuk a javítás lépését. Az iteratív lépés a 5. ábrán látható átalakítással egyezik meg:

```
minTime = get current welding time
FOR i FROM 0 TO seamCount-3
    FOR j FROM i+2 TO seamCount-1
        IF(minimalDistance < currentRouteDistance)
            swap i+1 and j seams
            reverse sublist(i+2, j-1)
            swap entry and exit in sublist(i+1, j)
            correct entry and exit points
            time = get swapped welding time
            IF(time < minTime)
                switch sequence
```

A kód igazat ad vissza, ha bármilyen változtatás történt, és hamisat, hogyha nem alakítottunk a szekvencián. Ennek fényében dől el, hogy folytatjuk-e javítást (igaz esetén), vagy pedig leállítjuk az algoritmust.

A használt *minimalDistance* változó az $i{+}1, j$ és az $i{+}2, j{+}1$ varratokhoz tartozó csonka kúppárok minimális távolsága. Ezeket a értékeket a címkézés alkalmával számítottuk ki minden párra a kúpokat befoglaló gömbökkel, és azok távolságával becsülve. A *currentRouteDistance* az aktuális túrában az $i+1, i+2$, valamint $j, j+1$ varratokhoz tartozó hegesztési szakaszok közti üresjárati robotmozgások összege.

A javító lépés leállításával a megoldásunk teljes. Megkaptuk a varratok sorrendjét, illetve minden korrekciós lépés alkalmával beállítottuk az adott illesztéshez tartozó hegesztési időt, ami a robot mozgási idejének, illetve a varraton töltött minimális hegesztési időnek a maximumaként állt elő. Ezután már lehetőségünk volt elemezni a kapott eredményeket, összehasonlítani azokat más algoritmusok adta megoldások hatékonyságával, futási idejével.

5.6. Kimenet

A széleskörű kiértékelhetőség biztosításához sok adatot kellett kimenetre vezetni. Minden futtatás során egy kimeneti fájl jött létre. Ez egy nagyobb adathalmazt tartalmazó, a varratútról, annak hosszáról, bejárási idejéről több információval rendelkező szövegfájl.

A fájlban egy kezdeti szekvenciát találhatunk, melyet a túraépítő algoritmus hozott létre. A javító iteráció lépéseinek fontosabb információi is megjelennek, ezekből következtethetünk a kezelés helyességére a különböző felbontású esetekben. Emellett megjelenik a hegesztési idő hossza, a teljes szekvencia valamint a program futásának ideje. Ez utóbbi optimalizálása is célunk volt a fejlesztés során így erre a paraméterre is szükségünk volt. A fájl neve a program bemeneteként megadható, a futtatási mappában jön létre.

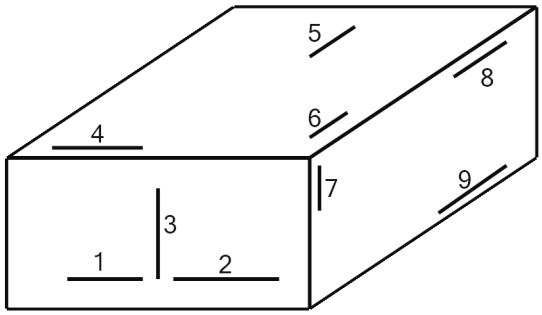
5.7. Alkalmazásfuttatás, tesztalanyok

A program használatához 3 érték megadása szükséges parancssori paraméterként:

- Bemeneti fájl neve: Egy, az osztályok felépítéséhez használt XSD-re validálható, XML fájl neve.
- Kimeneti fájl neve: A létrehozandó fájl neve, ahogyan az 5.6. szakaszban ismertettük.
- A hegesztési tér felbontása: A hegesztési pontok közötti koordinátánkénti távolság.

A tesztelés során több bementi fájlt is széleskörűen kiértékeltünk, majd ezeket a futási sebesség, túra hossza szempontjából elemeztük annak érdekében, hogy minél jobban pontosítani tudjuk az algoritmust.

Két típusú munkadarabbal dolgoztunk. Az egyik változatot a fejlesztés elején használtuk fel, amikor még gyors megoldásokra volt szükségünk, valamint arra, hogy ezek az eredmények egyszerűen ellenőrizhetőek legyenek. Az ilyen típusú tesztmunkadarabok egyszerű téglatestek, melyeken kevesebb varrat van annak érdekében, hogy a sorrend követhető legyen, és az algoritmusból kiszűrjük a nyilvánvalóan rossz részeket. Egy ilyen egyszerű munkadarabot mutat be, varratokkal együtt a 8. ábra.



8. ábra. Egyszerű típusú munkadarab

A másik típus, amelyet az algoritmus végső teszteléséhez használtunk fel, már több adatot tartalmaz és a munkadarab is egy igazi autóajtó mintájára lett elkészítve. Ezt már ismertettük, a 3. ábrán láthatunk egy ilyen elemet,

illetve a rajta elhelyezkedő varratokat. Ennek a segítségével az algoritmus hatékonyságát tudjuk ellenőrizni, illetve azt, hogy az egyszerűbb elemmel tesztelt modellünk megfelelően működik-e a nagyobb és komplexebb bemenetre is.

Több tesztelési folyamatot is végrehajtottunk, hiszen sok apró algoritmusunk van. Ezeknek mind különböző a hatása a két fontos paraméterünk, a hegesztési idő és a futási sebesség értékére. A következő fejezetben részletesen tárgyaljuk a módszereket és eredményeiket.

6. Tesztelés

A tesztelés során több bemeneti fájl - paraméter kombinációt is kipróbáltunk annak érdekében, hogy mind a hegesztési sebességet, mind pedig az algoritmus futási idejét optimalizáljuk. Több verziót is bemutatunk a programból, melyek a különböző sebességjavító algoritmusokat tartalmazzák. Össze fogjuk hasonlítani a kapott eredményeket két másik algoritmussal, melyek a miénktől eltérő megközelítéssel vágnak neki a problémának.

6.1. Verziótesztelés

A következő felsorolás az egyes verziók neveit, illetve a bennük található funkcionalitásokat tartalmazza. A verziók iteratívan növekednek, minden megtalálható bennük, amik az előzőek benne voltak.

- **1.0, Alap algoritmus:** A GTSP feladat megoldását tartalmazza, felépíti és javítja a túrát.
- **1.1, Minimal Distance (MD) algoritmus:** A javító algoritmus kiegészítését is tartalmazó verzió, melyben az 5.5. szakaszban leírt minimumtávolság feltételt is vizsgáljuk.
- **1.2, Split trim algoritmus:** Ebben a verzióban a korrekciós algoritmust alakítottuk át olyan módon, hogy az összes lehetséges pár helyett mindig az egyik pont lerögzítésével keressük az optimális be-, és kilépési pozíciókat.
- **1.3, Double split trim algoritmus:** A túra növelésének kiegészítése az előző verzió átalakításához hasonló ciklusbontással.
- **1.4, Welding point usage algoritmus:** Ez a legújabb verzió, melyben azt a többször ismertetett funkciót is megvalósítjuk, hogy a hegesztési pontok közül csak azokkal számolunk, amelyeknek a címkéje nem üres.

A teszteket egy 2,5 GHz-es Pentium(R) Dual Core E5200-as processzorral rendelkező, 4 GB belső memóriát, tartalmazó 64 bites Windows operációs rendszert futtató gépen késíztettük. Ennél erősebb hardverrel a számítási időigény tovább csökkenthető, és nagyobb felbontással a hegesztési időből is faraghatunk egy bizonyos határig.

Alap algoritmus

Ez a verzió csupán a sorrendtervezési feladat megoldását tartalmazza, nincs benne semmilyen más eszköz a sebesség növelésére. Ez a változat relatíve nagy futási idővel rendelkezik, így a kisebb bemeneti fájlokkal érdemes részletesebben tesztelni. A különböző felbontással készített futtatási eredményeket tartalmazza a 2. táblázat.

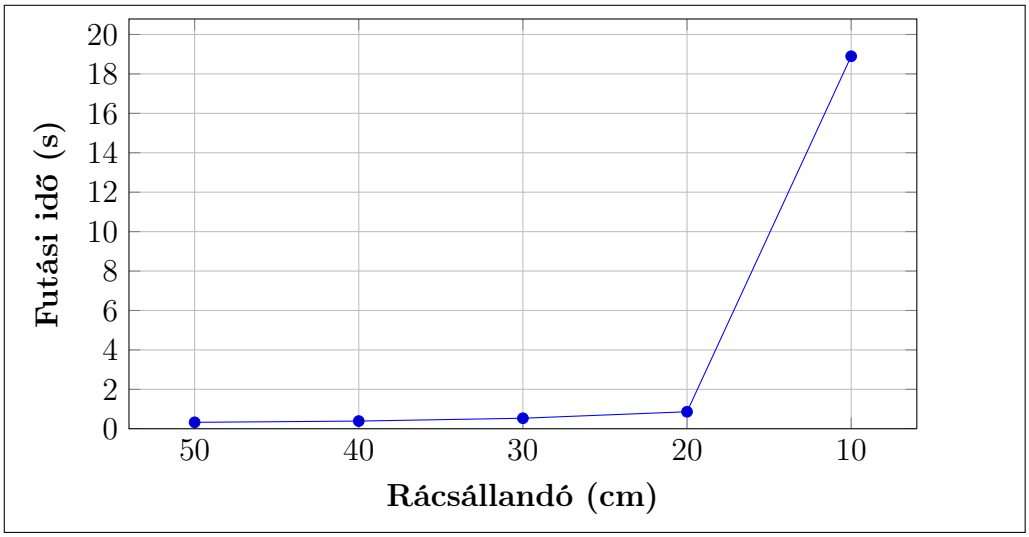
Bemeneti fájl	Rácsállandó	Hegesztési idő (s)	Futási idő (s)
minta1.xml	50	21,5917	0,327
minta1.xml	40	21,20971	0,39
minta1.xml	30	21,10728	0,534
minta1.xml	20	20,98	0,863
minta1.xml	10	20,93951	18,896
minta1.xml	5	20,91408	7503,84
minta3.xml	45	47,7372	0,362
minta3.xml	40	47,41171	0,416
minta3.xml	35	43,20278	0,525
minta3.xml	30	34,36711	0,701
minta3.xml	25	38,40628	0,635
minta3.xml	20	39,99853	1,053
minta3.xml	15	31,8741	3,925
minta3.xml	10	33,86369	86,412

2. táblázat. Az alap algoritmus teszteredményei.

Ebben a táblázatban látható, hogy a hegesztési idő a felbontás növelésével nagyrészt csökken. Ez el is várható, hiszen több lehetséges pont között kell

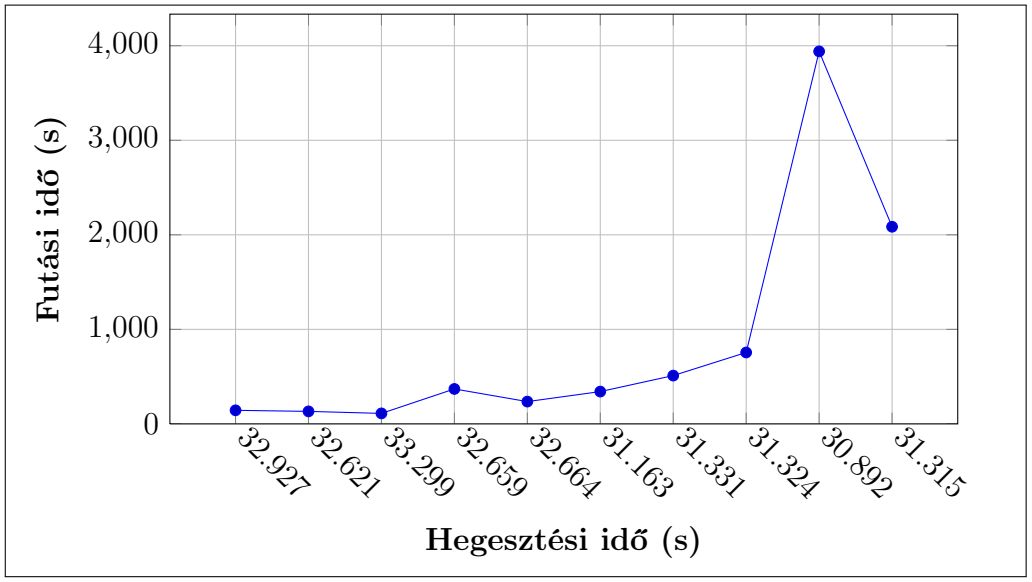
megtalálnia a megfelelő utat, sűrűbben helyezkednek el a pontok, ami szintén jó hatással van a hegesztési út bejárásához szükséges időre. Zölddel kiemelve láthatjuk az azonos tesztbemenethez tartozó leggyorsabb útvonal hosszát. Ez a 3. számú minta esetén nem a legkisebb felbontás esetén realizálódik. Ennek oka az lehet, hogy a hegymászó keresés során egy lokális optimumba ragadt az algoritmus a nagyobb felbontású esetben, vagy esetleg a legjobb felbontás olyan hegesztési teret alakított ki, amelyben lehetséges egy jobb útvonalat találni. A futási idő fordítottan arányos a rácsállandó nagyságával. Ez érthető, hiszen nagyobb felbontás esetén több hegesztési pontot kell megvizsgálnunk, több számítást kell végeznünk.

A minta1.xml bemenet esetén a 9. ábrán látható a rácsállandó és a futási idő kapcsolata (a láthatóság kedvéért a 7500 szekundumig tartó futtatást nem jelenítjük meg).



9. ábra. Teszteredmény a minta1 bemeneti fájlra

Ez verzió a nagyobb típusú fájlok esetén már elég nagy számítási idővel rendelkezik, így itt már jobban meg kell gondolni, hogy milyen rácsállandóval futtatjuk az alkalmazást, mert bár a hegesztési idő csökken a kisebb felbontás esetén de annak kiszámításához sokkal többet kell várnunk. A test1 komplex bemenetet vizsgálva, a hegesztési-, illetve futási idő kapcsolata a 10. ábrának megfelelően alakul.



10. ábra. Teszteredmény a test1 bemeneti fájlra

Látható, hogy 55-ös és 60-as rácsállandó esetében a program futás ideje már majd egy óra, amin mindenképpen csökkentenünk kellett, hogy az iparban is alkalmazható programot készítsünk.

MD algoritmus

Ez a verzió használja az 5.5. részben ismertetett algoritmust és ezzel csökkenti a javító lépés futásainak a számát. Ha az előző pontban leírt test1-re vett eredményekkel összevetjük, akkor megkapjuk, hogy a javítás az elvárásainknak megfelelően működött. Ez alatt azt értjük, hogy nem zártunk ki olyan lépést, mely javulást hozhatott volna, így ugyanazt a hegesztési utat sikerült megtalálnunk, mint az algoritmus nélkül. A fő haszna ennek a verziónak az előzővel szemben, hogy futási ideje jelentősen csökken.

A 3. táblázatban foglaljuk össze, hogy milyen mértékű ez a csökkenés a test1 bemeneti elem esetén.

Rácsállandó	1.0-s verzió (s)	1.1-s verzió (s)	Időarány (%)
100	143,68	48,72	33,9
95	132,49	55,19	41,65
90	111,10	59,22	53,3
85	369,80	144,43	39,05
80	236,06	117,65	49,83
75	341,88	173,01	50,6
70	511,06	253,17	49.53
65	755,54	395,47	52.34
60	3940,10	1004,15	25,48
55	2085,34	1151,45	55,21

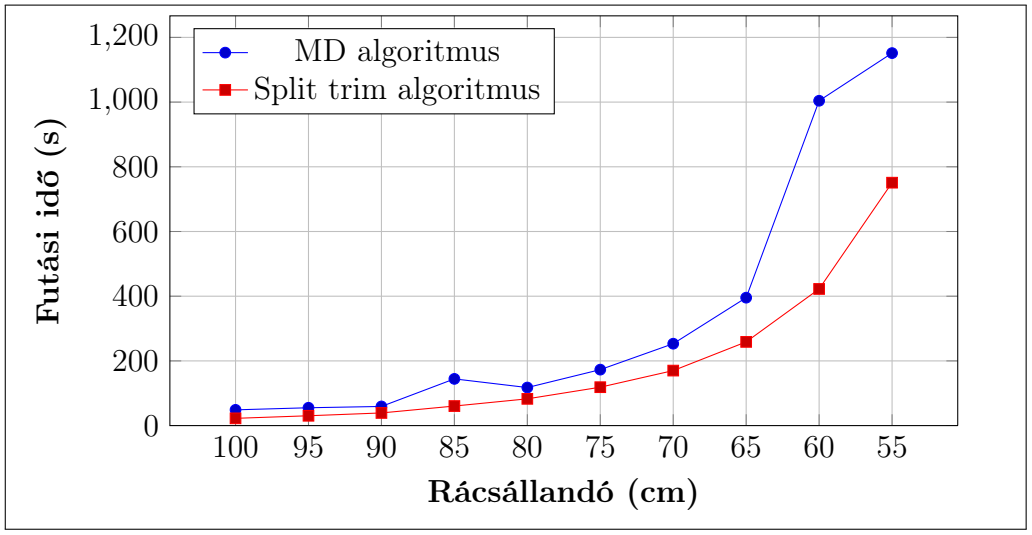
3. táblázat. 1.0-ás és 1.1-es algoritmus futási idejei

Látható, hogy 25-55%-ig terjed az idők aránya, ugyanez az intervallum a test2 bemenetre 39-52%-ra jön ki. Mindezek alapján sikeresnek tekintettük a javítást, és megtartottuk a változtatásokat. A későbbiekben az itt elért eredményekhez fogjuk hasonlítani az újabb algoritmusok által adott értékeket és azok szerint fogunk újabb következtetéseket levonni.

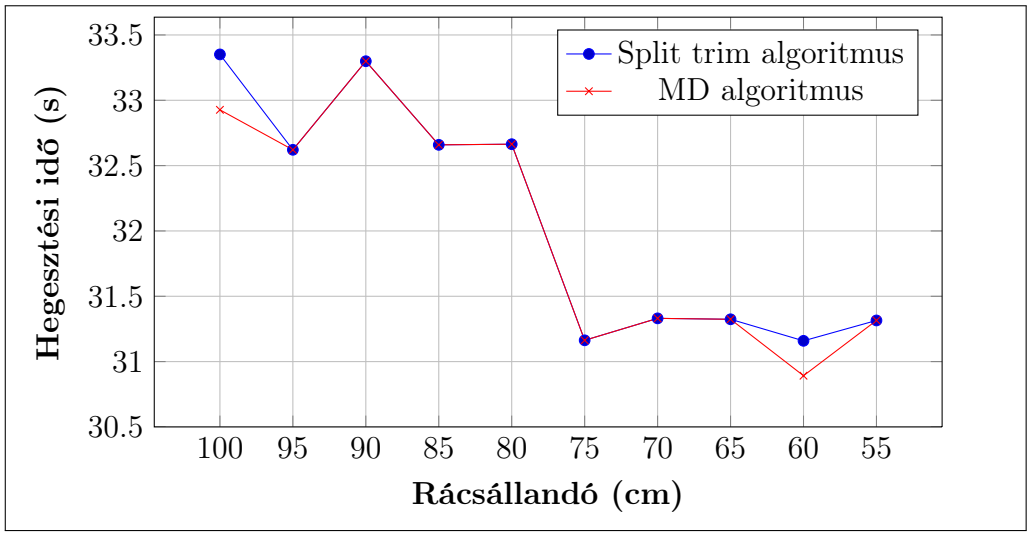
Split Trim

A következő javító lépés a javító algoritmus korrekciós algoritmusának átalakításával történt. Az eredeti koncepció szerint a legjobb be-, és kilépési pont megtalálása a lényeg minden esetben. Ha csökkentünk ezen az elváráson és előbb a belépési pont rögzítése mellett keressük a kilépésit, majd fordítva, akkor egy közelítő megoldáshoz juthatunk, emellett azonban a futási idő drasztikusan csökkenthet.

A 11. és 12. ábrák megmutatják, hogy a futási és hegesztési idők hogyan viszonyultak az előző pontban ismertetett MD verzió azonos eredményeihez. A teszteket itt a test1 bemeneten futtattuk az ábrákon látható felbontásokat vizsgálva.



11. ábra. 1.1-es és 1.2-es verzió futási idejének összehasonlítása



12. ábra. 1.1-es és 1.2-es verzió hegesztési idejének összehasonlítása

A megfelelő javulás ténye itt is vitathatatlan, legtöbb esetben ugyanazt az eredményt adta mint az MD és így mint az eredeti algoritmus, de futási ideje lényegesen gyorsabb volt. Ez pontos számokban kifejezve az előző szakaszban vett százalékokhoz hasonlóan kalkulálva 42-70%-ot jelent. Ezzel a kezdeti, kis adathalmazra is sok idővel operáló algoritmusunkat sikerült sokkal gyorsabbá tennünk.

A gyorsabb algoritmus lehetőséget teremtett nekünk arra, hogy a felbontáson csökkentsünk és így közelebb kerülhessünk az elméleti minimális

hegesztési időhöz. Ez előtt a kisebb felbontásoknak azért nem volt létjogosultságuk, mert túlságosan sok időt emésztett fel a kalkuláció, így hiába is adtak volna jobb eredményt, nem érte volna meg őket használni. Ez azonban megváltozott, jóllehet még így is elég hosszú ideig futottak. 50-es és 25-ös felbontásra próbáltuk ki ezzel a verzióval az algoritmust, ennek az eredményeit tartalmazza a 4. ábra. Látható, hogy 25-ös rácsállandóra még így is majd egy napig futott az algoritmus, vagyis a jobb hegesztési eredmény ellenére sokat kell várnunk a megoldásra.

Rácsállandó	**Hegesztési idő (s)**	**Futási idő (s)**
50	30,89424	1330,13
25	29,79036	82657,062

4. táblázat. 1.2-es verzió kisebb felbontásokkal test1 bemenetre

A hegesztési időt tehát 3.5%-al tudtuk javítani az által, hogy felére csökkentettük a felbontást, ez azonban a futási időt körülbelül 80-szorosára növelte. A következő verzióban további javításokat eszközöltünk és megpróbáltuk a sebességet növelni, hogy minél jobban közelíthessük az optimumot.

Double split trim

Az előző pontban ismertetett ciklusszétválasztó módszert nem csak a javítás során, de túra növelése közben is fel tudtuk használni. Amikor a legjobb indexet kerestük, akkor minden pontpárt megvizsgáltunk a varrathoz tartozó hegesztési térből. Ezt ismét kiválthattuk azzal, hogy egyszerre csak az egyik pontot mozgattuk.

Itt is hasonló eredményekre számítottunk, mint az előző pontban, és ezeket meg is kaptuk. Az 5. és 6. táblázatokban összegyűjtöttük azokat az eredményeket, amelyeket a program az 1.2-es és 1.3-as verziója adott, valamint összehasonlítottuk őket.

Rácsállandó	1.2-s verzió (s)	1.3-s verzió (s)	Javító arány (%)
100	33,351	32,483	2,604
95	32,621	32,726	-0,322
90	33,299	32,734	1,696
85	32,658	32,811	-0,466
80	32,664	32,286	1,156
75	31,163	31,389	-0,726
70	31,331	31,817	-1,550
65	31,324	31,287	0,117
60	31,159	31,668	-1,631
55	31,315	31,030	0,910
50	30,894	30,576	1,029
25	29,790	30,274	-1,624

5. táblázat. 1.2-es és 1.3-as verzió összehasonlítása a hegesztési idő alapján

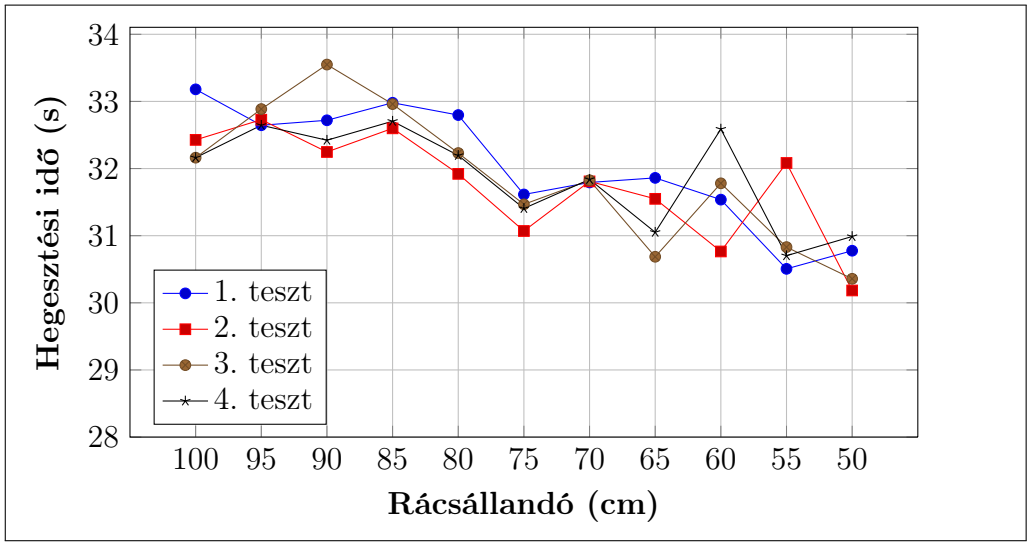
Látható, hogy a hegesztési idő néhány esetben rosszabb lett (pirossal jelölt esetekben). Ezek a visszaesések azonban elhanyagolható mértékűek ahhoz képest, hogy mennyit nyerünk a futási időben:

Rácsállandó	1.2-s verzió (s)	1.3-s verzió (s)	Időarány (%)
100	22,487	6,427	28,582
95	30,433	7,780	25,564
90	39,161	9,030	23,058
85	60,460	10,224	16,910
80	82,771	13,181	15,924
75	118,736	13,838	11,655
70	170,071	15,647	9,200
65	258,794	19,464	7,521
60	422,412	24,700	5,847
55	750,897	33,537	4,466
50	1330,130	42,785	3,217
25	82657,062	261,898	0,317

6. táblázat. 1.2-es és 1.3-as verzió összehasonlítása a futási idő alapján

Ezek alapján ismét egy olyan módszert találtunk, melynek segítségével a hegesztési idő kis mértékű növekedése mellett csökkenteni tudtuk az algoritmus lefutásának idejét egy olyan szintre, mely már kezelhető kis felbontású esetre is.

Szükségünk volt egy olyan véletlen generátorra, amely a varratoknál az első iterációban az addig nem létező kilépési pontot kiválasztja. Ennek fényében az volt az elvárásunk, hogy az algoritmusnak az a tulajdonsága, hogy két egymás utáni futtatásban ugyanazt az eredményt adja, elveszik. A feltevés megerősítésére többször egymás után lefuttattuk a programot ugyanazokra a bemenetekre és az eredményeket rögzítettük. A 13. ábrán láthatjuk, hogy az eredmények milyen mértékben különböznek a többszöri futtatás esetén.



13. ábra. 1.3-as verziós többszöri futtatása

Az alkalmazás tehát különböző megoldásokat adott az egymás utáni futtatásokban. Az eltérés 0-10%-ig mozog mind az első mind a második komplex tesztesetet figyelembe véve. Ez elég soknak tűnik azonban a felhasználó egyszerűen kezelheti ezt a problémát, ha többször lefuttatja a programot és a legjobbat választja ki az eredmények közül. Ezt megteheti, hiszen a futási idő törtrészére csökkent, ahogy a táblázatokból láthatjuk, így még néhányszoros időnövekedés sem okozhat problémát.

Welding point usage algoritmus

Az algoritmus tesztelése során arra lettünk figyelmesek, hogy nem használtunk ki egy, a rendszerben lévő lehetőséget az idő optimalizálására. Amint már többször említettük, az üres címkével rendelkező hegesztési pontokkal nem éri meg számolnunk. Ehhez át kellett alakítani utólag a programot, hogy az 5.3. szakaszban megadott pszeudo-kóddal megegyező funkcionalitást kapjunk. Mivel sokszor számolunk a tér pontjaival, így ez a változtatás is sebességcsökkenést okozott. Mivel az algoritmus a 1.3-as verzióban történt átalakítás következtében 10%-on belüli, nem előre meghatározható hegesztési időket szolgáltat, így ebben a tesztelési feladatban nem térünk ki a hegesztési időkre.

Az eredmények ismét azt támasztják alá, hogy a javító algoritmus sikeres volt. Az futási idők ismét csökkentek, 70-90% között van a javulás mértéke. Emellett a hegesztési idők nem változtak sokat, az átlagokat tekintve ±1-2% eltérés mutatkozott az előző pont átlagaihoz képest, amit csak a randomizált algoritmus okozhatott.

Ezzel az utolsó verzióval sikerült egy olyan változatot létrehozni, amelynek futási ideje az eredeti algoritmuséhoz képest több nagyságrenddel kevesebb, amellett, hogy a hegesztési időre vonatkozó hatékonysága megmarad. Számokban néhány felbontásra a sebességnövekedés nem is fejezhető ki, tekintve, hogy vannak olyan esetek, melyek túl sokáig tartottak volna az eredeti algoritmusnak, így nem indítottunk el velük a feldolgozást (például 25-ös rácsállandó). A 7. táblázatban néhány esetben összehasonlítottuk a kezdeti megoldás és az utolsó verzió eredményeit. Az 1.4-es algoritmusnak 4 esetre vonatkoztatott átlagai szereplenek a valós eredményeknek, tekintve, hogy itt is megmaradt az az 1.3-as verziótól a rendszerben lévő tulajdonság, hogy a végső eredmények egymáshoz képest 10%-on belül mozognak.

Rács-állandó	**1.0: Heg. idő (s)**	**1.4: Heg. idő (s)**	**1.0: Futási idő (s)**	**1.4: Futási idő (s)**
80	32,664	32,272	236,058	1,698
75	31,163	31,881	341,877	1,738
70	31,331	31,877	511,057	1,909
65	31,324	31,620	755,538	2,130

7. táblázat. 1.0-ás és 1.4-es verzió összehasonlítása

Láthatjuk, hogy a végső algoritmus eredménye a hegesztési sebességben nem sokban különbözik az alap algoritmus értékeitől, melyeknek kiszámításához sokkal több variációt néztünk végig. A valódi előnyt azonban nem ezekben az esetekben tapasztalhatjuk. A 4. táblázatban láthattuk, hogy 25-ös rácsállandóra a futási idő kis híján eléri az egy napot. Ezt már az 1.3-as verzióban sikerült pár percre csökkenteni, az utolsó verzióban ez tovább csökkent, és a program mindössze 30 másodperc körüli futási idővel végez a számításokkal. A javulás tehát vitathatatlan, nagy mértékben csökkenthetjük a rácsállandót, a futási idő úgy is kezelhető mértékű marad.

Kipróbáltuk a kisebb rácsállandójú esetet, de a program azokban az esetekben olyan sok memóriát emészt fel, melyet a tesztkörnyezetben használt gép már nem tudott kiszolgálni, így nem kaptunk eredményeket. Ez nem is csoda hiszen 25-ös rácsállandó esetén is már több mint egymillió rácspontunk van az 1. számú tesztmunkadarab használatakor. Ha öttel csökkentjük a hegesztési pontot távolságát, akkor számuk rögtön a kétszeresére nő, és ezt a fajta növekedést a tesztgépünk már nem bírta memóriával kiszolgálni.

A legrészletesebb felbontásunk tehát a 25-ös pontköz, ennél lejjebb a tesz-

telés során nem tudtunk menni. A legjobb eredmény, melyet a gyors algoritmus többszöri futtatásával a kaptunk, a 29.472 szekundumos hegesztési idő volt az test1 bemenetre.

6.2. Hatékonyságtesztelés

Rendelkezésünkre állt két alternatív megoldó algoritmus, amelyek más koncepció szerint vágnak neki a probléma megoldásának. Ezeknek a megoldásaival hasonlítottuk össze az algoritmusunk eredményeit, és vontuk le a következtetéseket a létjogosultságot illetően.

Az egyik megoldás a [10] cikkben megtalálható G. Reinhart-féle megközelítés. Ebből két verzió is elkészült, az egyik csupán a varratok geometriai helyzete alapján végzi el a sorrendtervezést, azoknak a paramétereivel számol a TSP megoldása során. Elkészült ennek egy módosított változata is, mely nem a varratok pozíciójával számol, hanem a hegesztési csonka kúpoknak a középpontját veszi alapul, így azoknak az adatoknak a segítségével számol, melyek valóban lényegesek a hegesztési út kialakításában.

A 8. táblázat a két algoritmus eredményét, az általunk használt legjobb hegesztési utat adó megoldást, valamint egy, a futási idejében a Reinhart-féle megközelítés módszereivel közel azonos futási idővel rendelkező, eredményünket tartalmazza.

Algoritmus	**Rács-állandó**	**Hegesztési idő (s)**	**Futási idő (s)**
Reinhart-féle algoritmus	-	94.661	0.125
Reinhart-féle módosított algoritmus	-	30.341	0.204
1.4-es verzió	25	29.848	26.165
1.4-es verzió	200	40.829	0.659

8. táblázat. Teljesítmény a Reinhart-féle algoritmusokkal szemben

A táblázatban látható, hogy az algoritmusunk a Reinhart-féle megoldáshoz képest jobban teljesít. Bár futási ideje hosszabb, de még így is elég rövid idő alatt elvégezhető, így majd 70%-os teljesítménynövekedés érhető el, alig fél perc alatt. A módosított algoritmus közel azonos eredményt szolgáltat kevesebb idő alatt, ennek kiterjeszthetősége azonban hiányt szenved, melyet a tisztán geometrikus módszerrel együtt ismertetünk.

A másik módszer a folytonos térben keresi a legjobb hegesztési útvonalat, és a GTSP megoldására alkalmazott lokális keresés operátorain belül, on-line módon végzi el a szükséges geometriai számításokat. Ez az algoritmus kihasználja, hogy minden varrat egy-egy csonka kúpból hegeszthető, ami az esetleges láthatósági problémák miatt csak az egyszerűbb geometriával rendelkező munkadaraboknál teljesül. A minimális hegesztési idő 27.638 szekundumnak adódott, amit rendszer 4 másodperces futás alatt kalkulált ki. Ennél a mi algoritmusunk 7%-al teljesít rosszabbul, ami a hegesztési időt illeti. A futás sebességében is jobb ez a módszer, hiszen közel 7-szer gyorsabban megtalálta az eredményt, mint a mi rendszerünk, így mindkét általunk vizsgált és optimalizálandó mérték tekintetében egy jobb módszert ad meg.

Ez utóbbi algoritmus megkérdőjelezheti a mi megoldásunknak a létjogosultságát, felveti a kérdést, hogy ha kisebb értéket kaphatunk mind a hegesztési-, mind a futási időre, akkor miért lenne szükség az általunk kínált módszerre. Amint már a 3.4. részben említettük, a láthatóságvizsgálat és ütközésdetekció fontos részei a technológiának és ha célunk az ipari alkalmazhatóság elérése, akkor elengedhetetlen számunkra, hogy figyelembe vegyük ezeket. A fentebbi algoritmus nem képes egyszerű módon figyelemben venni ezt a két elemet, míg a mi algoritmusunkban, ahogy már az előbb említett részben leírtuk, egyszerűen modellezhető lenne mind az ütközésvizsgálat, mind pedig a láthatóság ellenőrzése.

7. Összefoglalás

Az autóipari hegesztési technológia nagy átalakuláson ment keresztül. Megjelent a távoli lézer hegesztés, melynek sajátossága, hogy a robot a munkadarabtól távol helyezkedik el az műveletvégzés során. Az új módszer új problémákat vet fel. Meg kell határoznunk a varratok hegesztési sorrendjét és meg kell terveznünk a robot mozgási pályáját, hogy a hegesztés folyamata teljes mértékben automatizálható legyen, ami a sorozatgyártáshoz elengedhetetlen.

Célunk egy olyan algoritmus elkészítése volt, amely képes arra, hogy a technológia által adott bemeneti információk alapján egy közel optimális varrat sorrendet és pályát találjon meg a munkadarab körüli térben. Ezt az utat követve a robot képes a varratokat megfelelő minőségben minimális idő alatt elkészíteni.

Egy ilyen rendszer elkészítéséhez szükségünk volt egy matematikai modellre, amelyben a probléma megoldható. Sikeresen alkalmaztuk az utazó ügynök probléma egy általánosított változatát arra, hogy a sorrendtervezési feladatot modellezzük. A feladatot átvezettük erre a széleskörűen vizsgált,

és több megoldási módszerrel is rendelkező problémára. A lehetséges megoldások közül a [11] leírásban található kétfázisú módszert választottuk, ezt alkalmaztuk az általánosított modell megoldására.

Az elkészült megoldásunkat kisebb, általunk kreált elemeken, és az iparban használt ajtómodelleken is teszteltük, majd levontuk a megfelelő következtetéseket, hogy a programot javíthassuk. Sok időt fordítottuk a hegesztési időn kívül az algoritmus futási idejének optimalizálására, hogy az ipar által támasztott sebességigényeknek megfeleljünk és az alkalmazásunk széleskörűen felhasználható legyen.

A kapott eredményeket összevetettük két másik algoritmussal, amelyek egy-egy különböző megközelítés alapján dolgoznak. Az egyik korábbi algoritmusnál minden szempontból jobb eredményeket értünk el. A másik algoritmus, bár alacsonyabb hegesztési idővel rendelkező megoldásokat talál, felhasználhatósága az egyszerűbb geometriával rendelkező munkadarabokra korlátozódik. Ezzel szemben az itt bemutatott algoritmus könnyen általánosítható a komplex munkadarabokhoz szükséges láthatósági és ütközési korlátok figyelembevételével.

Ez alapján elmondható, hogy az elkészült program kezelhető idő alatt képes a műveleti sorrend, és a hegesztő robot pályájának kiszámítására. Ezen kívül egyszerűen kiegészíthető oly módon, hogy különböző, a felhasználás során fontos korlátokat betartson.

Hivatkozások

[1] `http://www.industrial-lasers.com/articles/2010/02/Remote-fiber-laser-welding-brazing-and-cutting.html`.

[2] `http://img.directindustry.com/images_di/photo-g/6-axis-arc-welding-robot-15481-2499159.jpg`.

[3] Arash Behzad and Mohammed Modarres. A new efficient transformation of generalized travelling salesman problem into travelling salesman problem. *Proceedings of the 15th International Conference of Systems Engineering (ICSE)*, 2002.

[4] D. Ben-Arich, G. Gutin, M. Penn, A. Yeo, and A. Zverovitch. Transformation of generalized atsp into atsp. *Operations Research Letters*, 31(1):357–356, 2003.

[5] Kenneth Castelino, Roshan D'Souza, and Paul K. Wright. Toolpath optimization for minimizing airtime during machining. *Journal of Manufactoring Systems*, 22(3):173–180, 2002.

[6] Miller Electric Mfg. Co. Handbook for resistance spot welding. `http://www.millerwelds.com/`, 2012.

[7] Kovács D. L. Intelligens rendszerek i. laboratórium - 1. laborgyakorlat segédlete. `www.mit.bme.hu`, 2010.

[8] Ed Ort and Bhakti Mehta. Java architecture for xml binding. `http://www.oracle.com/`, 2003.

[9] H-S. Park and H-W. Choi. Development of digital laser welding system for car side panels. In X. Na, editor, *Laser Welding*, pages 181–192. InTech, 2010.

[10] G. Reinhart, U. Munzert, and W. Vogl. A programming system for robot-based remote-laser-welding with conventional optics. *CIRP Annals – Manufacturing Technology*, 57(1):37–40, 2008.

[11] Király Tamás és Szegő László. Online jegyzet az egészértékű programozás i. és ii. tárgyhoz. 2012.

[12] KUKA Laser Solution. Ks roboscan remote 3d laser technology. `http://www.kuka-systems.com`.

[13] Comau S.p.A. Smartlaser – remote 3D laser welding. `http://www.comau.com`, 2012.

[14] Trumpf-Laser. Deep penetration. `www.us.trumpf.com`, 2012.

[15] G. Tsoukantas, K. Salonitis, A. Stournaras, P. Stavropoulos, and G. Chryssolouris. On optical design limitations of generalized two-mirror remote beam delivery laser systems: the case of remote welding. *The International Journal of Advanced Manufacturing Technology*, 32(9–10):932–941, 2007.